Teil 3

Anwendungs-entwicklung mit Lotus Domino

I

Einleitung

Der Domino Designer ist eine integrierte Entwicklungsumgebung, die zur Entwicklung von Anwendungen für Lotus Domino zur Verfügung steht. Grafische Werkzeuge, Dokumentenorientierung, Gruppenunterstützung, Workflow-Komponenten und die unterschiedlichsten Programmiersprachen sind Bestandteil dieses Werkzeuges. Ob visuelle Gestaltung von Masken oder Editierung von Programmtexten, die Entwickleroberfläche enthält Hilfsmittel zu ihrer Durchführung. Der Vorteil dieser Integration besteht darin, dass die Wirkung und das Aussehen von Gestaltungselementen

sofort überprüft und wieder angepasst werden können. Die Syntaxprüfung des Editors unterstützt alle vom Designer verwendeten Programmiersprachen und erleichtert die Erstellung von fehlerfreien Anwendungen.

Für die Entwicklung unter Lotus Domino stehen mehrere Sprachen zur Verfügung, mit denen auf die in Lotus Domino enthaltenen Datenbankfunktionen zugegriffen werden kann. Die einfachste Form davon sind die „Einfachen Aktionen“, die bei Ereignissen hinterlegt und in Agenten (Programmen) verwendet werden können.

Bei den Programmiersprachen ist die Formelsprache als Basis zu sehen, die wichtige Steuerungsfunktionen enthält, die von keiner der anderen Sprachen in Lotus Domino abgedeckt werden. Mit ihrer Hilfe kann, ähnlich wie man es z.B. von Lotus 1-2-3 oder Excel her kennt, auf Felder und andere Gestaltungselemente zugegriffen werden. Die Formelsprache bietet allerdings nur begrenzte Strukturierungshilfsmittel und kann nicht für jeden Zweck eingesetzt werden.

LotusScript wurde als zusätzliches Programmier-Tool erst später eingeführt. LotusScript ist ein BASIC-Dialekt, der etwa dem Stand von Visual Basic 6.0 entspricht. Die Sprache wurde um spezielle Notes-Klassen und Objekte erweitert. LotusScript ist auch in anderen Anwendungen von Lotus, wie z.B. in Word Pro verfügbar. Als weitere Alternative können Java oder JavaScript für die Entwicklung verwendet werden. Welche Sprache die „richtige“ ist, hängt von der Anwendung ab. Nicht für jede Situation stehen in den einzelnen Sprachen Elemente zur Verfügung. Deshalb müssen auch alle Sprachen in der Domino-Entwicklung berücksichtigt werden.

1.1 Planung und Konzeption

Vor eine saubere Anwendungsentwicklung gehört, unabhängig von der Plattform, eine genaue Analyse und Definition dessen, was als Lösung gewünscht ist. Erst wenn ein eindeutiges Pflichtenheft für die Aufgabenstellung vorliegt, ist es möglich, eine passende Anwendung zu entwickeln. Lotus Domino ist sicher eine starke Entwicklungsplattform. Es ist aber nicht unbedingt für jede Problemstellung der günstigste Lösungsweg. Sollen z.B. große Mengen relationaler Daten sequentiell oder über Join verbunden, abgearbeitet werden, so finden sich dazu sicher günstigere Werkzeuge als gerade Lotus Domino. Auf der anderen Seite ist es mit Hilfe von Lotus Domino ohne weiteres möglich, relationale Datenbanken direkt zu bearbeiten.

Soll mit Lotus Domino entwickelt werden, so sind die Datenbanken zu definieren, außerdem die Objekte sowie deren Eigenschaften und Beziehungen. Meist ist die Hardwareplattform vorgegeben. Bei Lotus Domino ist die Auswahl schwieriger, da sowohl auf der Server-Seite als auch bei den Clients eine Auswahl möglich ist. Alle wichtigen Server-Systeme einschließlich der Mainframes können eingesetzt werden. Bei den Clients sind auch Systeme außerhalb der Windows-Welt verfügbar.

Eine Inhouse-Anwendung kann als reine Domino-Anwendung konzipiert werden. Immer häufiger soll aber auch der Einsatz im Browser möglich sein. Je nach Einsatzumgebung müssen unterschiedliche Programmiertechniken ausgewählt werden.

Je komplexer die Anwendung werden soll, desto wichtiger ist eine klare Strukturierung der Daten. Die einzelnen Datenbanken

sollten nur zusammengehörige Daten enthalten. Im Prinzip entspricht eine Datenbank einer Datei mit verschiedenen Indizes oder einem zusammenhängenden Teilprogramm.

Das äußere Erscheinungsbild einer Software wird immer wichtiger. Durch die Verwendung von Bildern, animierten Grafiken und anderen optischen Hilfsmitteln wird ihr Einsatz für den Benutzer attraktiver, sie belastet den Entwickler aber auch mit zusätzlichen Designproblemen. Neben den reinen Funktionen müssen grafische Grundsätze beachtet werden. Während der Designphase müssen Framesets, Gliederungen, Masken und Ansichten so definiert werden, dass sowohl Funktionalität als auch Erscheinungsbild in unterschiedlichen Laufzeitumgebungen optimiert sind.

Und es wird natürlich auch noch programmiert. Die Anwendungsplanung muss die Funktionen mit den Anwendern abstimmen und in Programmiervorgaben umsetzen. Bei Lotus Domino ergibt sich jetzt zusätzlich die Qual der Auswahl einer Programmiersprache. Viele Funktionen sind auf verschiedenen Wegen realisierbar. Leider kann es aber auch vorkommen, dass nur eine Sprachvariante einsetzbar ist, weil nur sie die gewünschte Funktionalität bietet.

Ist die Anwendung fertiggestellt, so beginnt das umfangreiche Testen. Nicht nur Programmierer, sondern auch Anwender müssen diesen Teil der Anwendungserstellung unterstützen. Erst wenn eine fehlerfreie Nutzung der Software gewährleistet ist, kann der Start für den praktischen Einsatz gegeben werden.

Dann ist der Zeitpunkt gekommen, um die Software zu verteilen und die spätere Aktualisierung zu planen. Ohne ein genaues Update-Konzept sollte keine Software ausgeliefert werden.

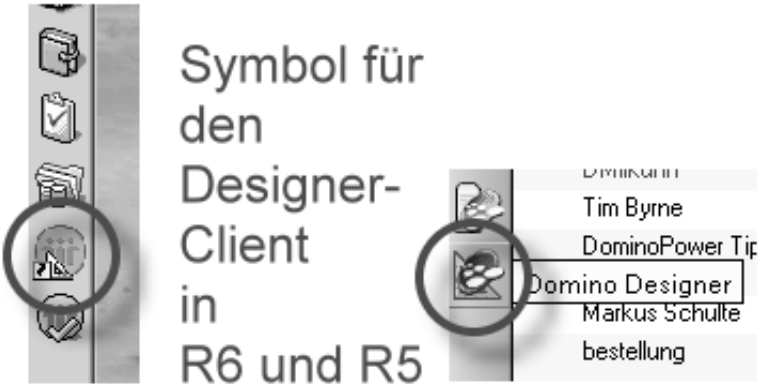
2

Der Domino Designer

Seit der Version 5 von Lotus Domino wird der Designer als eigenständiges Programm zusätzlich zum normalen Client geliefert. Der Domino-Designer enthält die Werkzeuge zur Gestaltung von Masken, Ansichten und Ordnern. Er beinhaltet aber auch eine Entwicklungsumgebung für verschiedene Programmiersprachen.

Februar 2003 / 12. EL

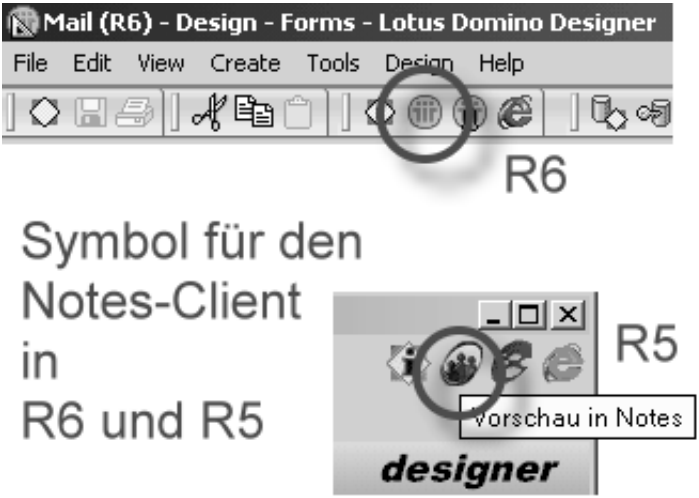
Der Designer-Client kann als eigenes Programm direkt aus dem Startmenü, aus einem Symbol auf dem Desktop oder aus dem Notes-Client heraus gestartet werden:



Lesezeichen für den Domino-Designer

Das Lesezeichen wird bei der Installation in der Lesezeichenleiste angelegt.

Umgekehrt startet der Designer-Client zur Anzeige eines Gestaltungselementes automatisch den Notes-Client. Er verfügt gleich über zwei Symbole zum Starten des Notes-Clients: eines in der Lesezeichenleiste und eines bei den Navigationsschaltflächen im Kopfbereich des Designer-Fensters.



Lesezeichen für den Notes-Client

Das Designer-Modul ist an eine eigene Lizenz gebunden und muss entsprechend eigenständig installiert werden. Nur Arbeitsstationen, bei denen dieses Programm installiert wurde, können deshalb auch den Designer-Client aufrufen.

Diese Seite ist zur Zeit nicht belegt.

2.1 Die Installation des Designer-Clients

Im Gegensatz zu früheren Versionen muss der Designer-Client getrennt installiert werden. Die Berechtigung, den Designer zu verwenden, ist technisch nicht mehr von der beim Benutzer eingetragenen Lizenzart abhängig. Bei der Installation der Clients wird entschieden, ob der Designer-Client auf dem jeweiligen Arbeitsplatz installiert werden soll. Ist das der Fall, muss man bei der Installation „Alle Clients“ anklicken. Das Programm wird dann zusammen mit dem Notes-Client und dem Administrator-Client installiert. Unter „Programme / Lotus-Anwendungen“ befindet sich nach der Installation ein zusätzlicher Menüpunkt: „Lotus Domino Designer“.

Diese Seite ist zur Zeit nicht belegt.

2.2 Die Benutzeroberfläche des Designers in Release 5

Nach dem Starten des Designer-Clients wird man mit einem Auswahlschirm begrüßt, von welchem man aus entweder direkt eine neue Datenbank anlegen oder eine bestehende Datenbank zur Bearbeitung öffnen kann. Daneben sind Links zu einigen Internet-Seiten von Lotus Notes zum Thema „Entwicklung unter Lotus Notes“ angeordnet, über die man sich aktuelle Informationen von Lotus für Entwickler suchen kann.

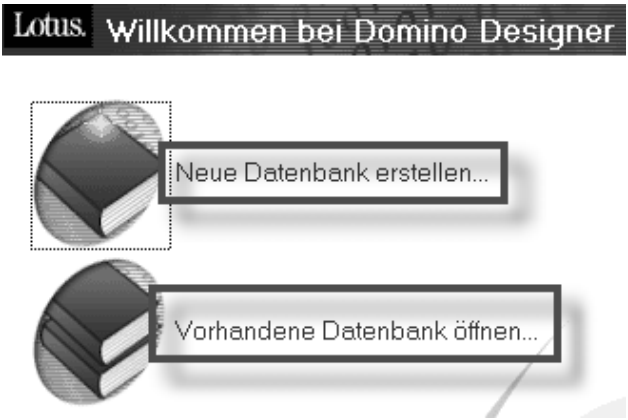
Das Willkommenfenster des Designers

Das Hauptfenster des Domino-Designers teilt sich in mehrere Bereiche auf: den Menübereich, die SmartIcons, die Register für geöffnete Fenster, die Lesezeichenleiste und die eigentliche Arbeitsfläche.

Das Menü zeigt nach dem Start lediglich die Positionen „Datei“, „Bearbeiten“ und „?“ an. Erst nach dem Öffnen von Gestaltungselementen wird das Menü durch die Positionen „Ansicht“, „Erstellen“ und „Aktionen“ angepasst.

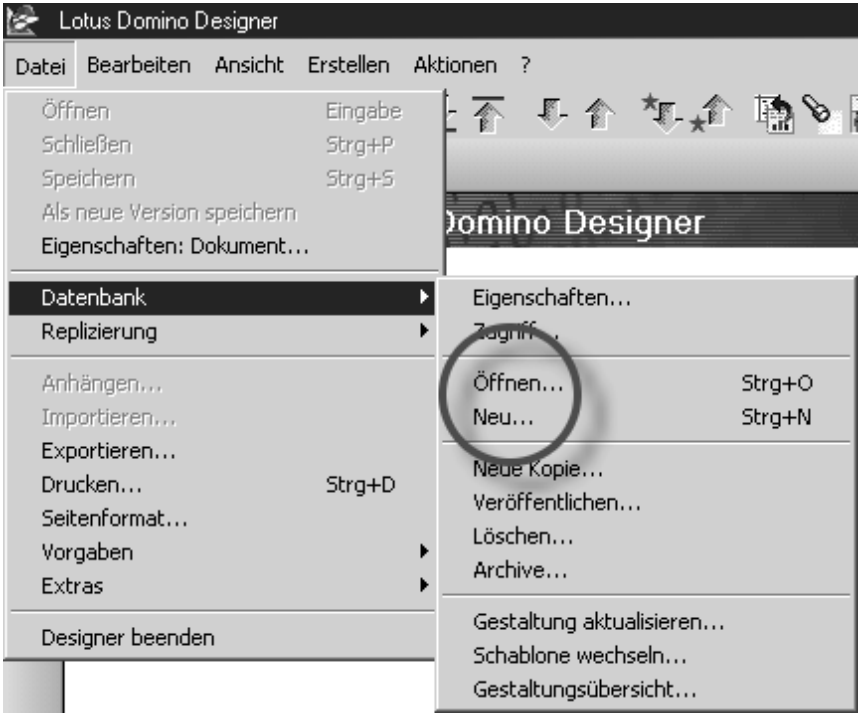
Das Menü nach dem Start

Der Einstieg in die Bearbeitung erfolgt entweder über die Symbole auf der Willkommenseite „Neue Datenbank erstellen“ bzw. „Vorhandene Datenbank öffnen“ oder über das Menü „Datei“ mit den Positionen „Datenbank / Öffnen“ bzw. „Datenbank / Neu“.



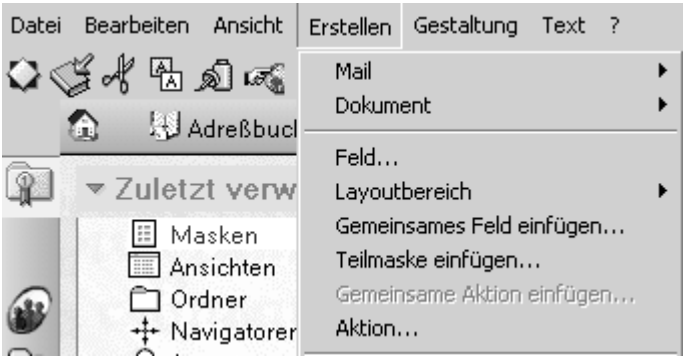
Datenbanksymbole in dem Willkommenfenster

Wird eine neue Datenbank ausgewählt, so erscheint ein Dialogfenster, in das weitere Angaben zu der neuen Datenbank eingetragen werden. Wird „Datenbank / Öffnen“ verwendet, so erwartet der Designer im folgenden Dialog den Namen und den Speicherort der gesuchten Datenbank.



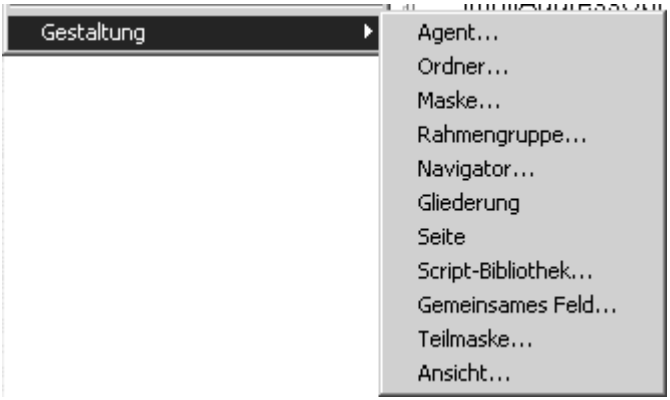
Das Menü „Datei – Datenbank“

Die für die Bearbeitung von Gestaltungselementen wichtigen Menüpositionen finden sich unterhalb der Position „Erstellen“.



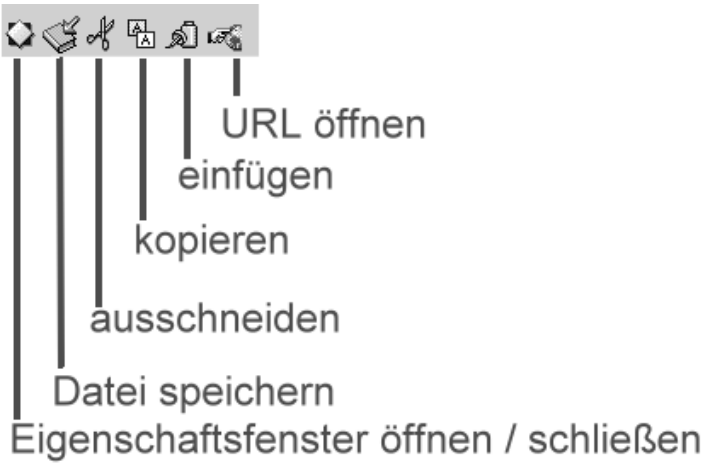
Das Menü „Erstellen“

Dort und in der Unterposition „Gestaltung“ befinden sich alle Gestaltungselemente, die in den Datenbanken zur Entwicklung von Anwendungen eingesetzt werden.



Die Position „Gestaltung“

Die SmartIcons werden einblendet, wenn das in dem Menü „Datei / Vorgaben / SmartIcons“ aktiviert wurde. Generell wird eine Reihe von Standardsymbolen angezeigt.



Die Standardsymbole

Zusätzliche Symbole werden in Abhängigkeit von der jeweiligen Bearbeitungssituation verwendbar. Alle Symbole können individuell im Vorgaben-Menü bearbeitet, gelöscht oder ergänzt werden. Die genaue Vorgehensweise dazu ist im Teil 1 des Handbuches beschrieben.

Im rechten Teil am oberen Rand sind die Navigationsschaltflächen zu sehen. Sie erlauben es, folgende Aktionen auszulösen:

- Eigenschaftsbox ein- oder ausschalten
- Vorschau über den Lotus-Domino-Server (nur bei aktiver Serververbindung möglich)

- Anzeige im Microsoft Internet Browser
- Anzeige über den Netscape Communicator

Navigationsschaltflächen

Diese Aktionen können auch über das Menü „Gestaltung“ ausgelöst werden.

Wie beim Notes-Client findet sich am linken Fensterrand die Lesezeichenleiste. In dieser Leiste können Lesezeichen und Lesezeichenordner gespeichert werden. Wird ein Lesezeichen angeklickt, so öffnet sich die entsprechende Datenbank. Wird ein Lesezeichenordner geöffnet, so erscheint eine Lesezeichenseite mit Lesezeichen.

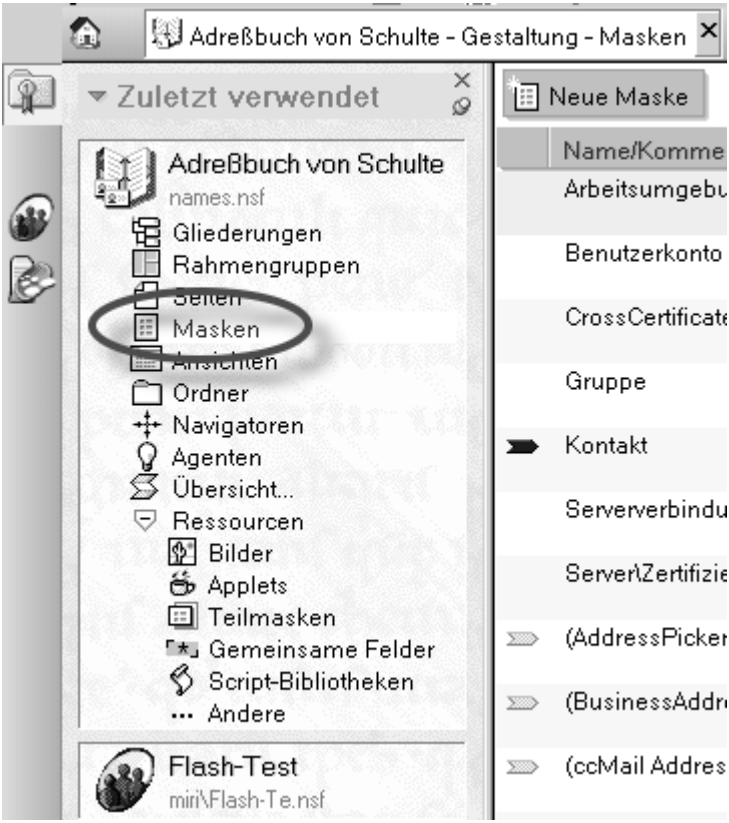
Das generelle Lesezeichen „Zuletzt verwendet“ öffnet einen Lesezeichenordner mit allen Datenbanken, die zuletzt bearbeitet wurden und die nicht aus diesem Lesezeichenordner entfernt wurden.

Es ist als Hilfe gedacht, dass jeweils die zuletzt geöffnete Datenbank als oberste angezeigt wird. Arbeitet man allerdings mit mehreren Datenbanken im Wechsel, so ist es reichlich verwirrend, wenn sich ständig die Reihenfolge der geöffneten Datenbanken im Lesezeichenordner ändert.

Der Lesezeichenordner „Zuletzt verwendet“

Besser ist es deshalb, eigene Lesezeichenordner auf der Lesezeichenleiste anzulegen, die zusammengehörige Datenbanken aufnehmen.

Jede Datenbank kann kompakt oder mit ihren Gestaltungsgruppen geöffnet angezeigt werden.



Gestaltungselemente einer Datenbank

Folgende Gruppen von Gestaltungselementen können ausgewählt werden:

- **Gliederung**
  Auswahlmenüs im Explorer-Stil

- **Rahmengruppen**
  Aus der Web-Seitenentwicklung bekannte „Framesets“ und „Frames“, die dazu dienen, andere Gestaltungselemente aufzunehmen und zu gruppieren.

- **Seiten**
  Anzeigeelemente, die keine Felder zur Eingabe von Daten enthalten.

- **Masken**

- **Ansichten und Ordner**
  Dokumentlisten

- **Navigatoren**
  grafische Elemente zur Steuerung von Ansichten

- **Agenten**
  Programme

- **Übersicht**
  Hilfsmittel zur Auswertung der Gestaltungselemente einer Datenbank

- **Ressourcen**
  weitere Gestaltungselemente

Unter der Position Ressourcen verbergen sich weitere Gestaltungselemente:

- **Bilder**
  Grafikdateien, die in die Datenbank eingebunden und für Gestaltungszwecke in Seiten und Masken verwendet werden.

- **Applets**
  Java-Programme, die als Bausteine in der Anwendung eingesetzt werden.

- **Teilmasken**
  Masken, die alle üblichen Maskenelemente enthalten können und die in andere Masken eingebunden werden.

- **Gemeinsame Felder**
  Felder, die einmal definiert und mit allen ihren Eigenschaften bei unterschiedlichen Masken zum Einsatz kommen.

- **Scriptbibliotheken**
  Sammlungen von LotusScript-Prozeduren und Funktionen, die in Gestaltungselementen verwendet werden.

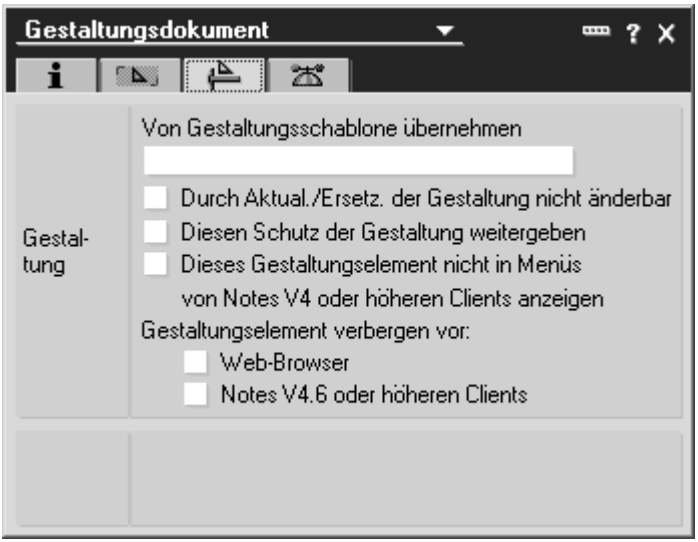
- **... Andere**
  Datenbanksymbol, das Dokument „Benutzen dieser Datenbank“, das Dokument „Über diese Datenbank“, Datenbank-Script und Gemeinsame Aktionen

Die Gestaltungsgruppen führen dann im Entwurfsbereich zu der Auflistung aller Gestaltungselemente dieser Gruppe. Wenn z.B. über die Lesezeichen der Bereich der Masken ausgewählt wurde, dann kann danach im Entwurfsbereich mit Doppelklicken eine bestehende Maske zur Bearbeitung ausgewählt oder eine neue Maske angelegt werden.

Der Entwurfsbereich

In der Liste werden der Name, eventuelle Alias-Namen und Kommentare gezeigt. Programme können die Objekte über Alias-Namen verwenden. Damit sind sie weniger anfällig gegen Änderungen (z.B. durch Sprachversionen).

Die Eigenschaften des Objektes werden im „Eigenschaftsfenster“ definiert. Je nach Objekt ändert sich der Inhalt dieses Fensters.

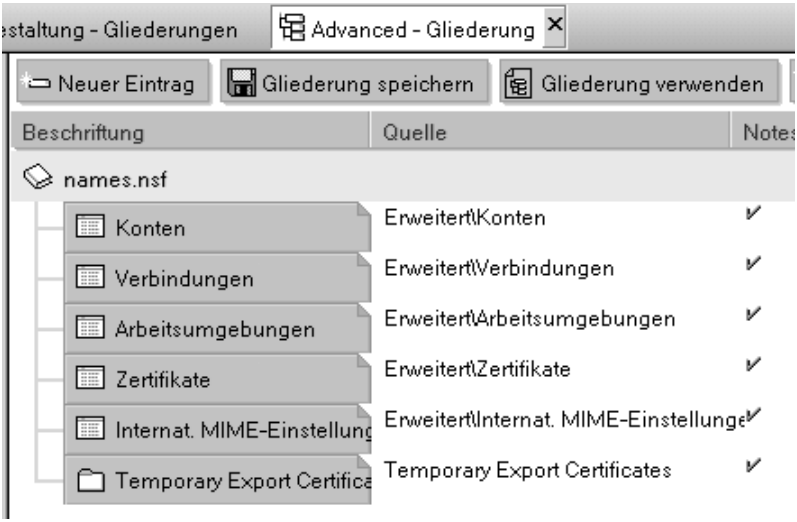


Eigenschaften der Gestaltungselemente

Wird ein Gestaltungselement angeklickt oder über die Schaltfläche „Neu“ ein neues erstellt, so öffnet sich jetzt der dazugehörige Editor.

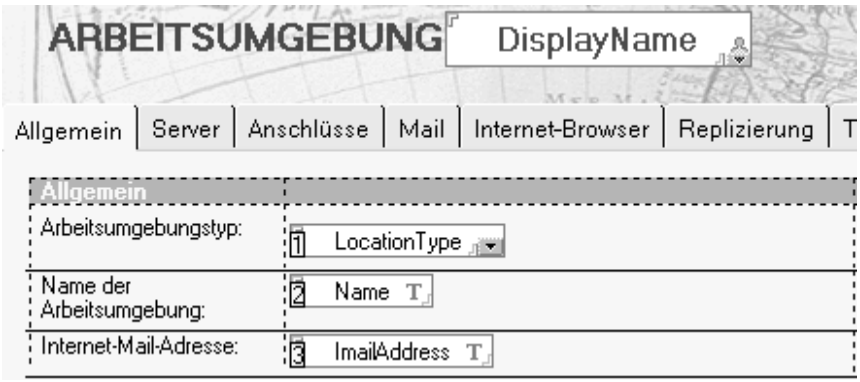
Der Designer verfügt über eine ganze Reihe solcher Editoren und erlaubt es damit dem Benutzer, auf einfache und schnelle Weise seine Gestaltungswünsche umzusetzen.

So zeigt sich zum Beispiel die Entwicklungsumgebung für Gliederungen in einer ganz anderen Art als die für Masken.



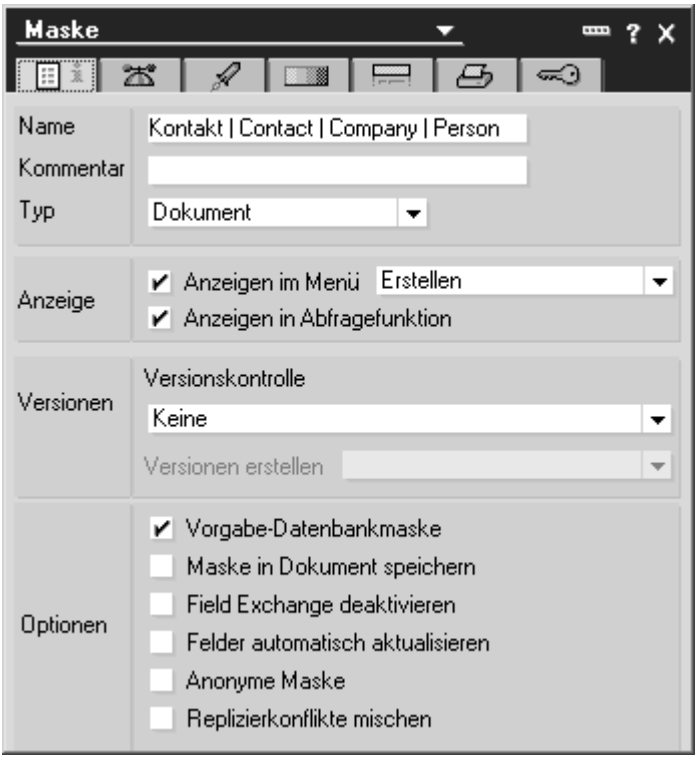
Der Gliederungseditor

Außerdem unterscheidet sich der Editor für Rahmengruppen von dem für Ansichten und Ordner.



Der Maskeneditor

Sobald ein Gestaltungselement geöffnet ist, ändert sich die zugehörige Eigenschaftsbox und es werden andere Register mit Inhalten angezeigt, die auf die Art des Gestaltungselementes zugeschnitten sind.



Eigenschaften einer Maske

Der Designer wird auf den üblichen Wegen geschlossen. Ist der Notes-Client ebenfalls geöffnet (z.B. zur Anzeige eines Gestaltungselementes), so muss er getrennt geschlossen werden.

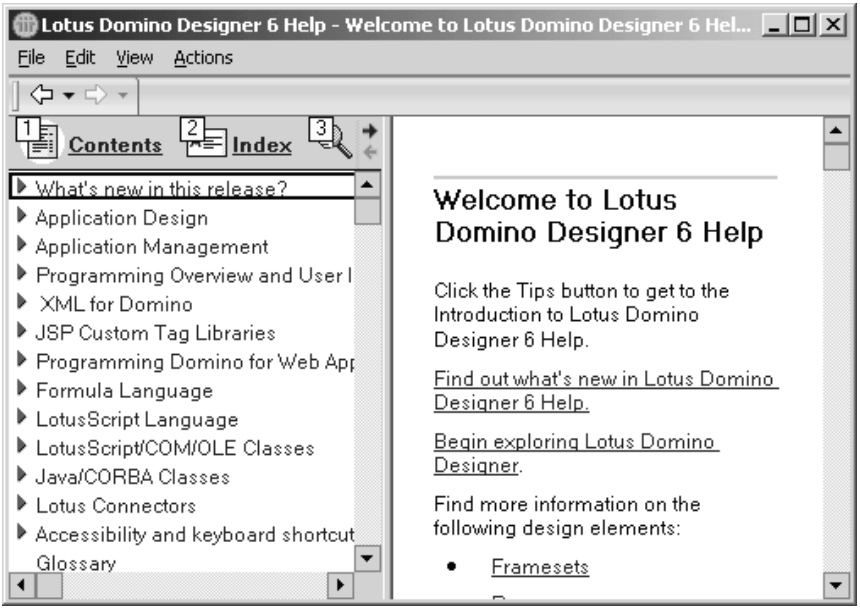
2.3 Die Benutzeroberfläche des Designers in Release 6

Wird der Designer-Client von Release 6 gestartet, sieht er seinem Vorgänger recht ähnlich. Die schon aus Release 5 bekannte Oberfläche wurde nur in wenigen Punkten verändert.

Standard-Datenbanken

Hilfsmittel auf der Startseite

Auch wenn sich das Aussehen geändert hat, so werden doch die gleichen Funktionen – nämlich das Erstellen einer neuen oder das Öffnen einer bestehenden Datenbank – angeboten. Als Unterstützung für den Entwickler werden URLs zum Internet und die Domino-Designer-Hilfe angezeigt.



Was ist neu in Release 6

Nachdem Lotus-Seiten im Internet nicht mehr geführt werden, verweisen die Verknüpfungen jetzt auf die entsprechenden IBM-Seiten. Die Startseite der Lotus Developer Domain (LDD) ist gespickt mit Verweisen zu allem, was im Zusammenhang mit Lotus benötigt wird. Neben den Produktbeschreibungen findet der Besucher dieser Seite:

- **Foren**
  Diskussionsforen zu den verschiedenen Lotus-Produkten

- **Aktuelle Artikel unter dem Titel „LDD today“**

- **Bibliothek der Dokumentationen**
  Artikel, Programmbeschreibungen, Redbooks, Whitepapers

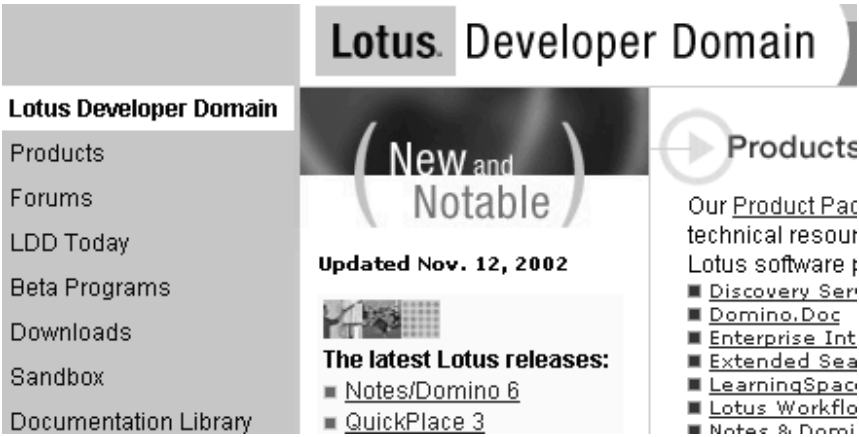
- **Beta-Programme**
  Beta-Versionen der einzelnen Lotusprodukte können geladen, getestet und diskutiert werden.

- **Downloads**
  Download-Seite mit aktuellen Produktversionen, Updates und Supportfiles.

- **Sandbox**
  Beispiele und Demonstrations-Datenbanken

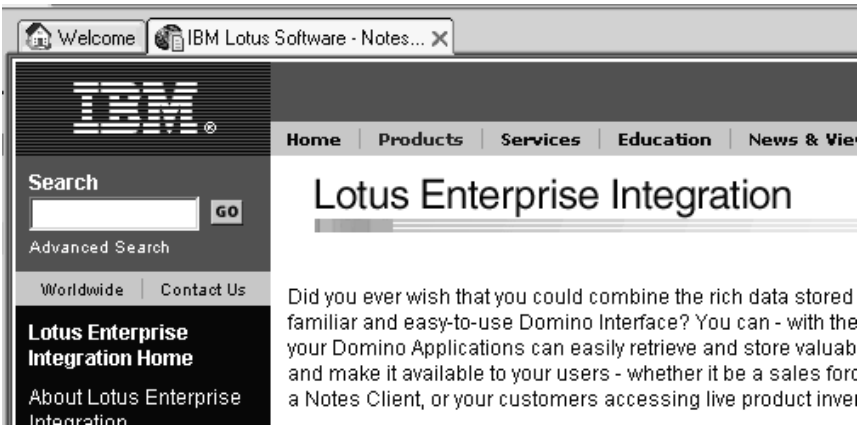
- **Spezielle Angebote**

Die Entwicklerseite LDD wird ständig aktualisiert und ist für einen Entwickler, der IBM-Produkte einsetzt, unentbehrlich.



LDD-Seite von IBM

Warum gerade die Seite für den LEI (Lotus Enterprise Integrator) an dieser Stelle benötigt wird, ist nicht ganz zu verstehen. Ein klein wenig Werbung muss schließlich auch sein...



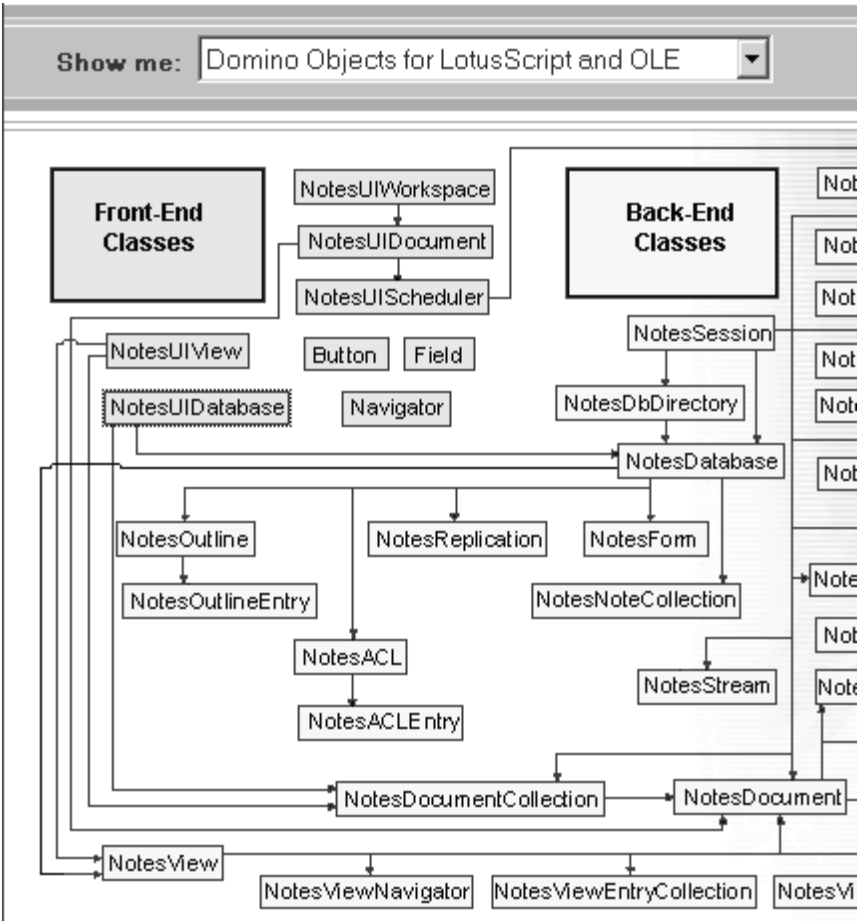
LEI-Seite von IBM

Etwas unscheinbar wurde ein schönes Hilfsmittel hinter einem Auswahlfeld mit der Überschrift „Allgemeine Aufgaben“ verborgen. Wird dieses Feld angeklickt erscheinen zusätzlich die Auswahlmöglichkeiten:

- **Domino-Objekte für LotusScript und OLE**
- **Domino-Objekte für DXL-Support**
- **Javascript-Objektmodell**

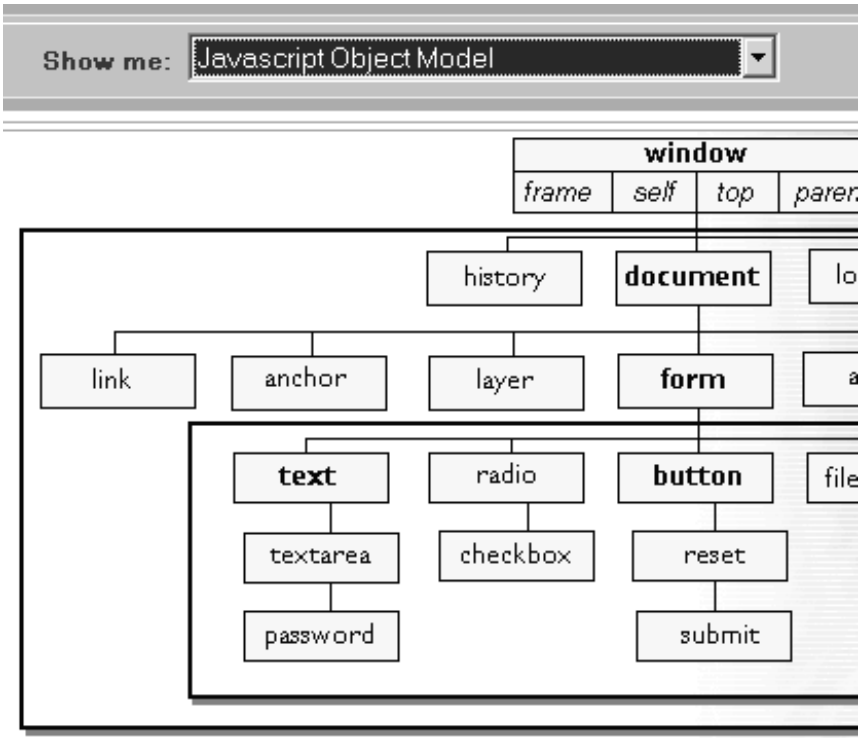
Jede dieser Auswahlen lässt ein Bild erscheinen, das dem jeweiligen Objektmodell entspricht.

Wer schon einmal versucht hat, die Zusammenhänge zwischen den verschiedenen Notes-Klassen von LotusScript zu verstehen, der wird dieses Schaubild zu schätzen wissen. Auch wenn es nicht alle Feinheiten zeigt, so verschafft es doch einen hervorragenden ersten Überblick.



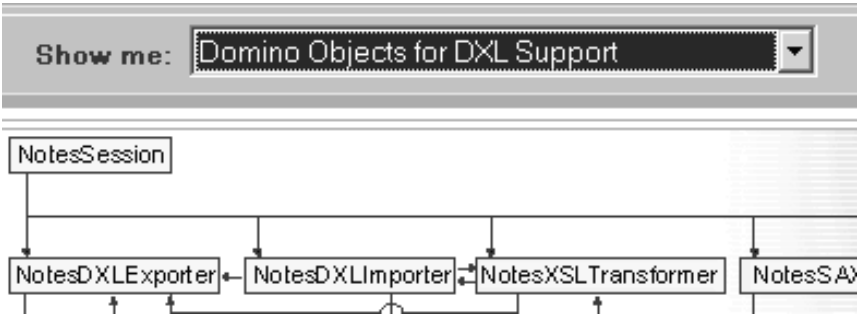
Das LotusScript-Objektmodell

Wer sich mit Javascript bereits auseinander gesetzt hat, dem wird das Schema des Javascript-Objektmodells von Lotus Domino sehr bekannt vorkommen. Es handelt sich um reine Standards, bei denen lediglich die Frage zu stellen ist, was wird im Notes-Client und was wird nur über die Browser-Anzeige unterstützt.



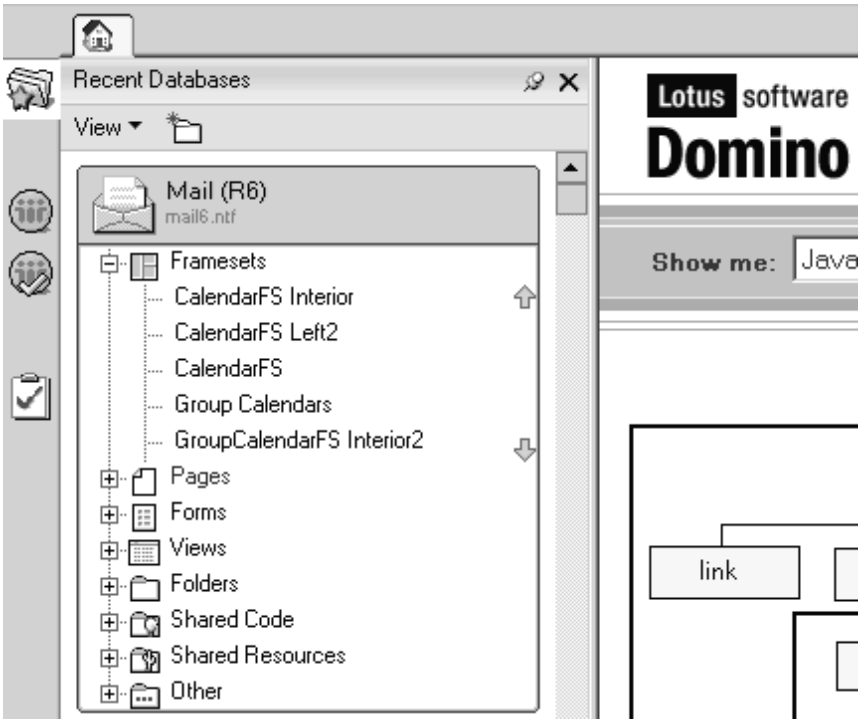
Das Javascript-Objektmodell

Ein ganz wesentliches neues Element im Release 6 stellt die „Umfangreiche Unterstützung von XML“ dar. Deshalb wird auch dafür ein Schema angeboten.



Das DXL-Objektmodell

Bei den Lesezeichen fällt zunächst nur auf, dass sie neue Symbole enthalten. Wie schon im Vorgänger-Release werden die geöffneten Datenbanken in einer Lesezeichenseite angezeigt. Erst wenn man ein Datenbankfenster öffnet, zeigt sich, dass hier in erheblichem Umfang Änderungen und Erweiterungen vorgenommen wurden.

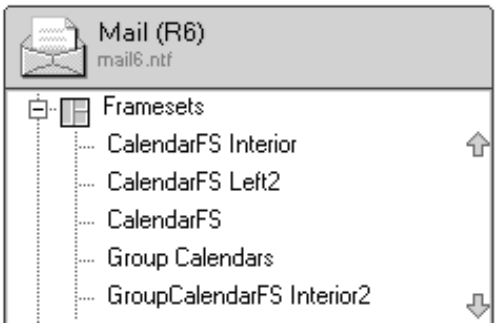


Die Gestaltungselemente einer Datenbank

Die Gestaltungselemente wurden zu folgenden Gruppen zusammengefasst:

- **Rahmengruppen**
  Rahmengruppen und Rahmen
- **Seiten**
  Informationsseiten ohne Bearbeitungsfelder
- **Masken**
- **Ansichten**
- **Ordner**
- **Gemeinsamer Code**
  Agenten, Gliederungen, Teilmasken, gemeinsame Felder, gemeinsame Aktionen und Script-Bibliotheken
- **Gemeinsame Ressourcen**
  Bilder, Dateien, Applets, Cascaded Style Sheets und Datenquellen
- **Sonstiges**
  Datenbank-Ressourcen, Navigatoren und Übersichten

Die Gruppen können im Explorerstil erweitert werden. Sie zeigen dann darunter eine Untergruppe oder fünf Zeilen der zugehörigen Liste von Gestaltungselementen. Es kann entweder die Liste auf der rechten Seite angezeigt oder mit Hilfe der kleinen Pfeiltasten die Anzeige durch die Liste gescrollt werden.

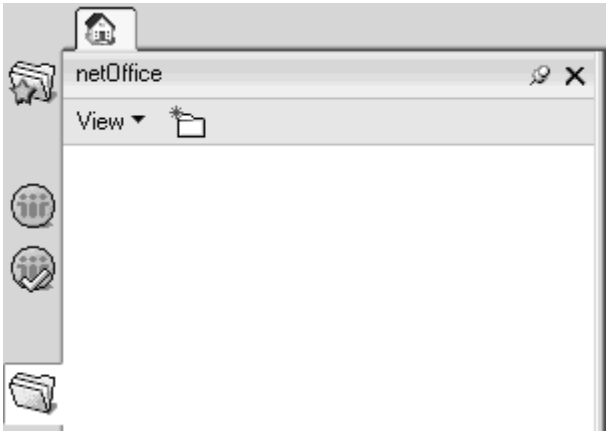


Ausschnitt aus den Gestaltungselementen

Ein Gestaltungselement kann durch Anklicken geöffnet werden, egal ob es in der Teilliste, in der Datenbankanzeige oder in der Gesamtliste der Gestaltungselemente im rechten Entwurfsfenster erscheint.

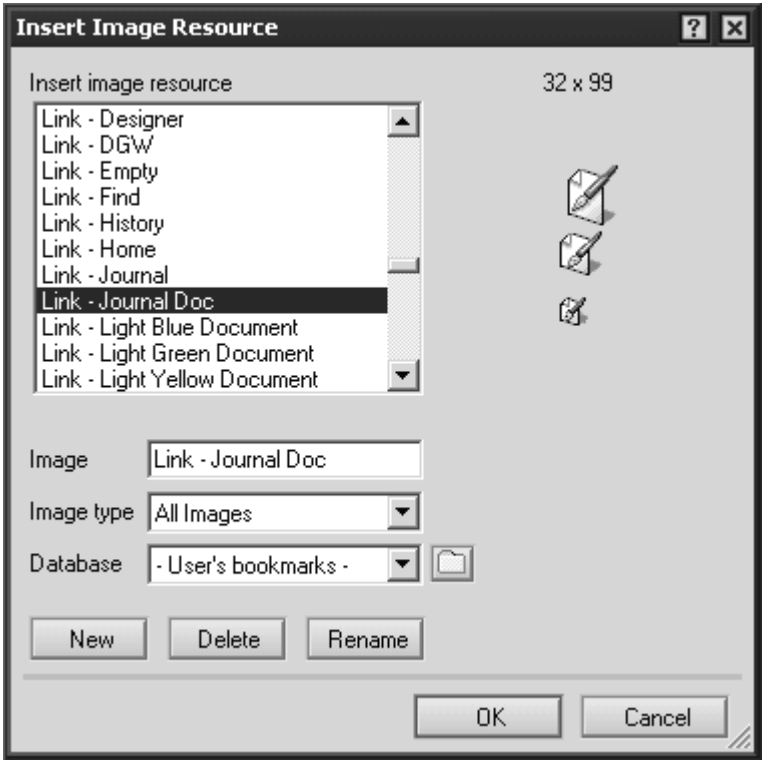
Neben der Standard-Lesezeichenseite können auch eigene Lesezeichenseiten und Lesezeichenordner angelegt werden. Dazu wird das Ordnersymbol im oberen Bereich einer Lesezeichenseite verwendet.

Symbol für neuen Lesezeichenordner



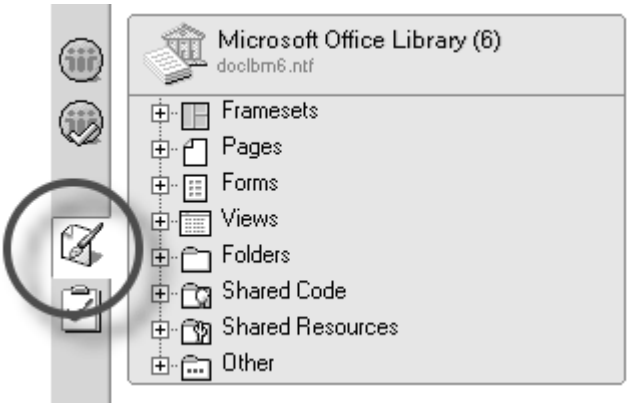
Neue Lesezeichenseite

Für die neue Lesezeichenseite kann ein eigenes Symbol gewählt werden.



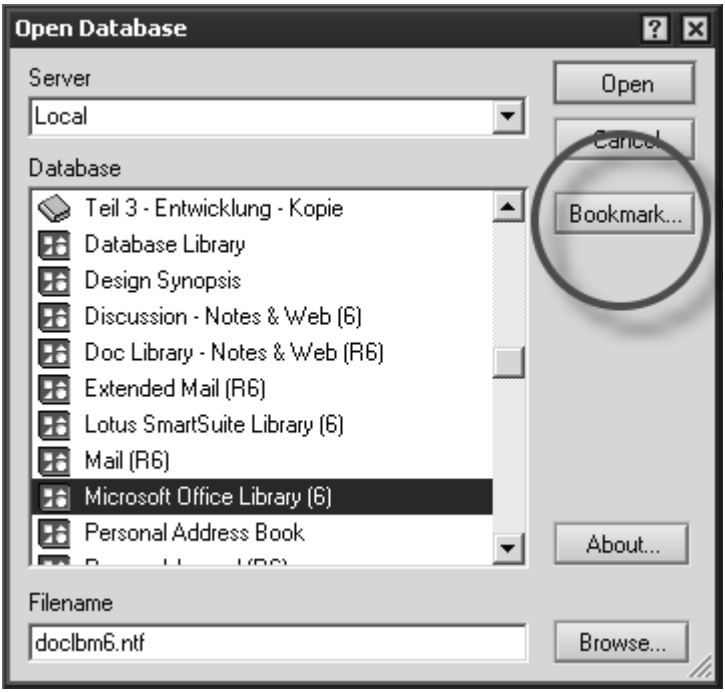
Symbol der Lesezeichenseite ändern

Spätestens beim Neustart des Designer-Clients sollte das neue Symbol in der Lesezeichenseite erscheinen. Für das Symbol eines Ordners innerhalb einer Lesezeichenseite kann das Symbol auf dem gleichen Weg verändert werden. Die Datenbanken zeigen das Symbol, das in der Gestaltung der Datenbank hinterlegt ist.



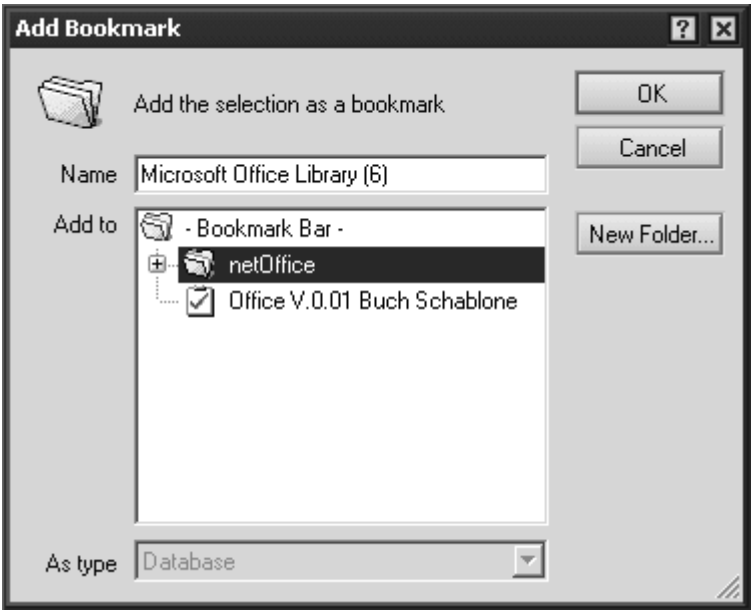
Neues Symbol der Lesezeichenseite

Um eine neue Datenbank zu einer Lesezeichenseite hinzuzufügen, wird über die Menüposition „Datei / Datenbank / Öffnen“ der entsprechende Dialog geöffnet.



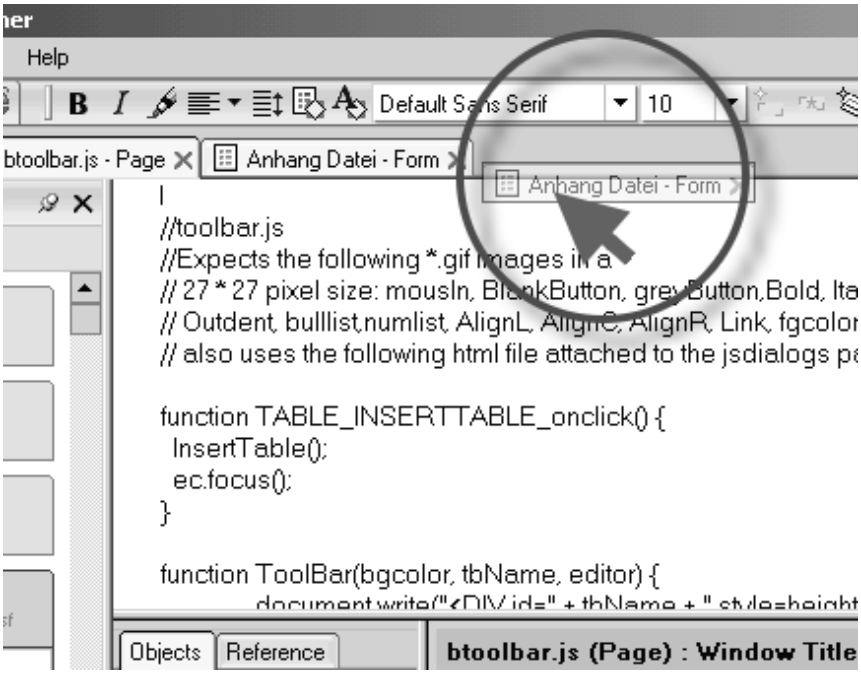
Datenbank öffnen

Über die Schaltfläche „Lesezeichen“ wird der Dialog gestartet, bei dem Name und Anordnung der neuen Datenbank festgelegt wird.



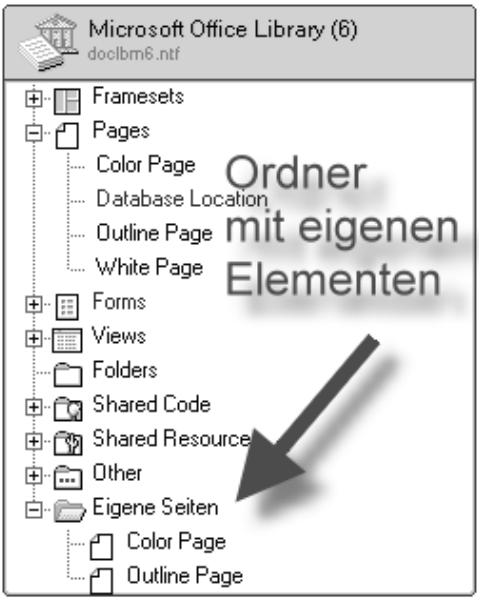
Datenbank zur Lesezeichenseite hinzufügen

Ist die neue Datenbank geöffnet, so wird sie in dem Entwurfsbereich als Fenster mit einem Register angezeigt. Dieses bereits in Release 5 verwendete Prinzip erleichtert das Umschalten zwischen verschiedenen Arbeitsbereichen. Wird das Register mit der Maustaste angeklickt und festgehalten, kann über „Drag and Drop" eine Verknüpfung zur dazugehörigen Datenbank in der Lesezeichenleiste erzeugt werden. Neu in Release 6 ist, dass die Register per „Drag and Drop" untereinander verschoben und damit in eine neue Reihenfolge gebracht werden können.



Register-Reihenfolge ändern

Eine weitere Neuerung in Release 6 findet man bei der Untergliederung von Gestaltungselementen einer Datenbank. Neben den festgelegten Gruppen können jetzt am Ende der Liste eigene Ordner für solche Elemente angelegt werden.



Eigener Ordner für Gestaltungselemente

Achtung!

Diese Möglichkeit kann nur innerhalb einer eigenen Lesezeichenseite verwendet werden. Weder in der Lesezeichenleiste noch in der Lesezeichenseite „Zuletzt verwendet“ sind solche Ordner möglich.

Ein solcher Ordner wird wie ein Lesezeichenordner mit dem entsprechenden Symbol am oberen Rand der Seite erstellt. Diese Ordner können z.B. für einzelne Teams verwendet werden, die nur bestimmte Elemente bearbeiten sollen. Denkbar ist auch, dass Elemente, die häufig bearbeitet werden, in solche Ordner ausgelagert werden.

❶❷❸ Ausführungsschritte:

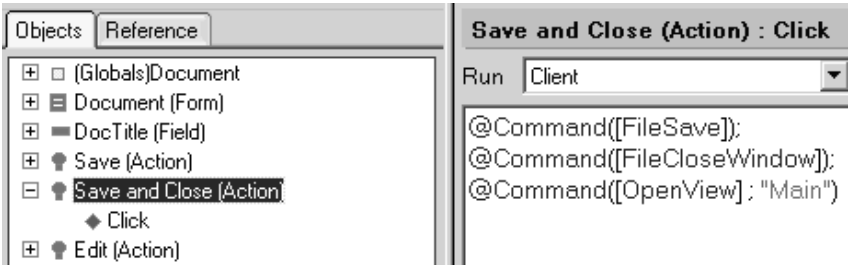
1. Vorhandene Lesezeichenseite öffnen oder neue Lesezeichenseite anlegen.
2. In der Lesezeichenseite eine Datenbank anlegen, falls sie dort noch nicht vorhanden ist.
3. Das Symbol für neuen Ordner am oberen Rand der Lesezeichenseite anklicken.
4. Im Dialog „Ordner erstellen“ den gewünschten Namen eintragen und die Datenbank auswählen, in der dieser neue Ordner erstellt werden soll.
5. Den Vorgang mit „OK“ abschließen.
6. Unter der Gestaltungsgruppe „...Andere“ erscheint der neue Ordner.
7. Mit „Drag and Drop“ werden Gestaltungselemente aus den anderen Gruppen in den neuen Ordner gezogen.

❶❷❸

Zur Bearbeitung der Gestaltungselemente steht der Entwurfsbereich bereit. Er ist unterteilt in den eigentlichen Gestaltungsbereich, der vom jeweiligen Editor ausgefüllt wird, den Objektbereich und den Bereich zur Aufnahme des eigentlichen Programmtextes. Alle Bereiche arbeiten eng zusammen und sind vom Objektzusammenhang her gesteuert. Wird z.B. eine Rahmengruppe entwickelt, ist nur der Gestaltungsbereich verfügbar. Wird eine Maske gestaltet, so sind alle drei Bereiche verfügbar.

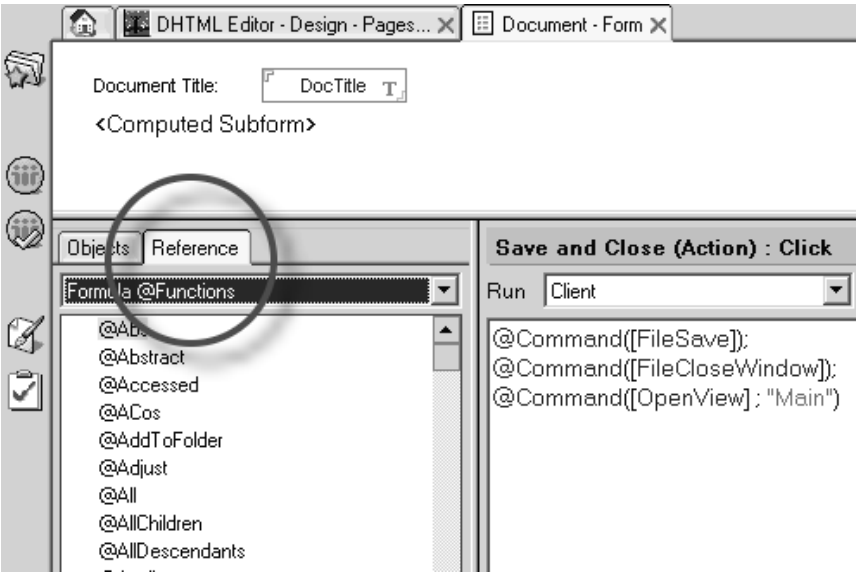
Der Entwurfsbereich

Der Objektbereich ist unterteilt über zwei Register am oberen Rand. Im Register „Objekt“ werden alle Objekte des aktuellen Gestaltungselementes angezeigt. Zu jedem Objekt können dessen Ereignisse angezeigt werden. Bei den Ereignissen kann dann im Bereich des Programmtextes ein Script hinterlegt werden, das mit dem Objektereignis gekoppelt ist. Ist ein solcher Programmtext vorhanden, wird das Symbol vor dem Ereignis farbig gekennzeichnet.



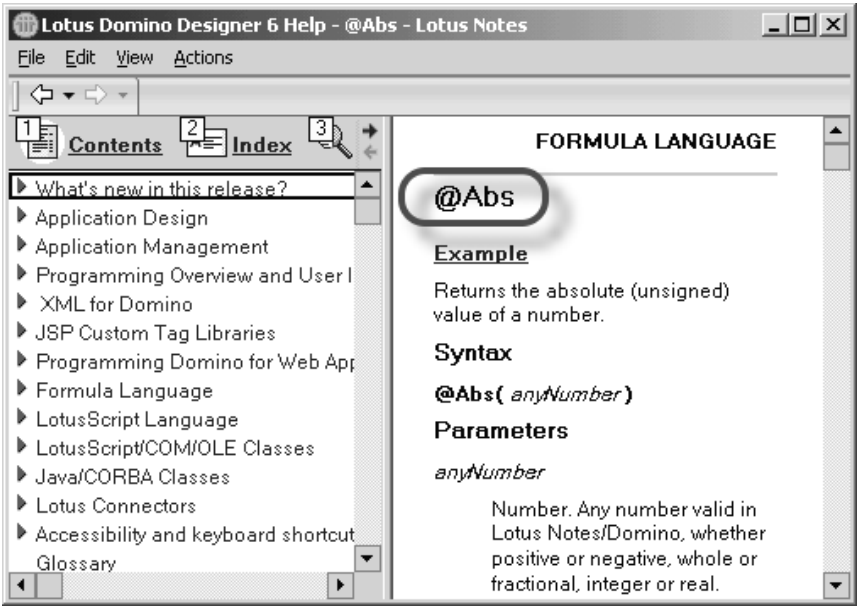
Objekte und Ereignisse

Im Register „Referenz“ sind alle verfügbaren Dokumentationen aufgeführt, die als Hilfeunterstützung verwendet werden können. Welche Handbücher angezeigt werden, hängt ganz von dem gerade ausgewählten Objekt und seinem Ereignis ab.



Hilfeunterstützung über Referenzen

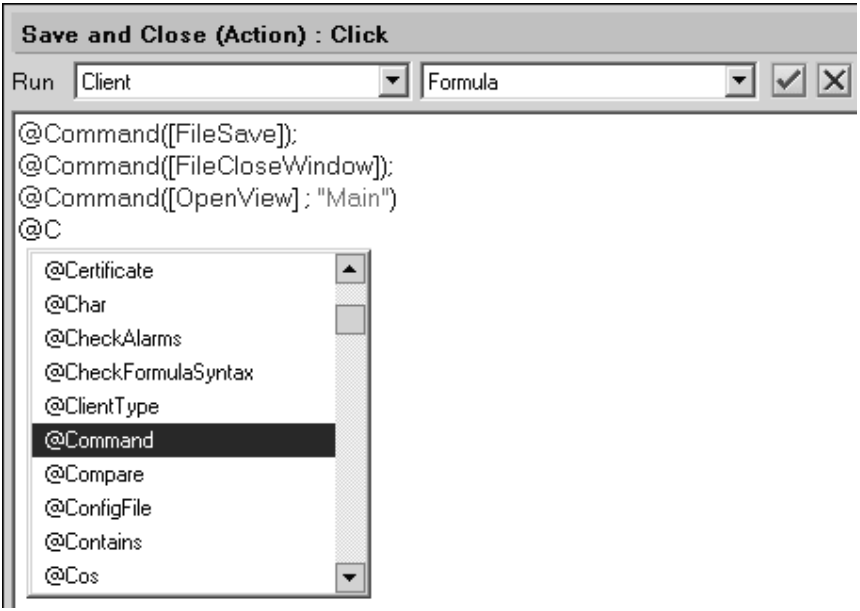
Wird eine Formel, ein Befehl, eine Funktion oder eine Methode ausgewählt, kann entweder der Text direkt im rechten Bereich eingefügt oder aber über eine Schaltfläche der Text der Hilfedatenbank gestartet werden.



Text in der Hilfedatenbank

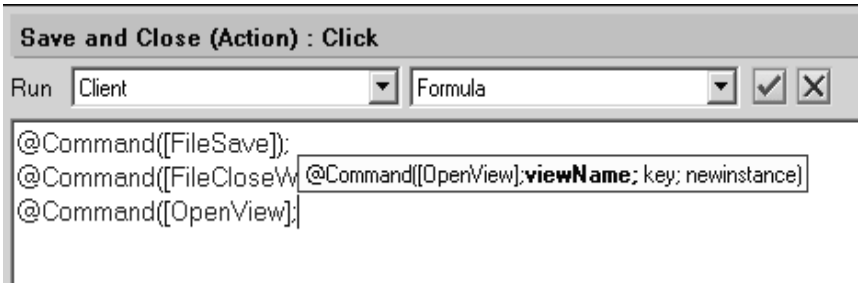
Damit bekommt der Entwickler ein Hilfsmittel an die Hand, das es ihm erlaubt, jederzeit die Einzelheiten zu bestimmten Programmfunktionen zu klären.

Wer bereits mit Entwicklungswerkzeugen von Microsoft gearbeitet hat, der kennt die Ergänzungsfunktion des Programmeditors. Wird ein Wort im Programmtext eingefügt, so bietet der Editor automatisch alle dazu passenden Schlüsselwörter der entsprechenden Sprache an. Diese Funktion hat IBM mit Release 6 auch dem Lotus-Domino-Designer spendiert. Sobald damit begonnen wird, einen Text zu erfassen, öffnet sich eine Auswahlliste, in der alle im Zusammenhang möglichen Schlüsselwörter auf geführt sind. Ist das richtige Wort bereits markiert, reicht das Betätigen der Enter-Taste. Soll ein anderes verwendet werden, kann es über die Pfeiltaste ausgewählt werden.



Automatische Schlüsselwortanzeige

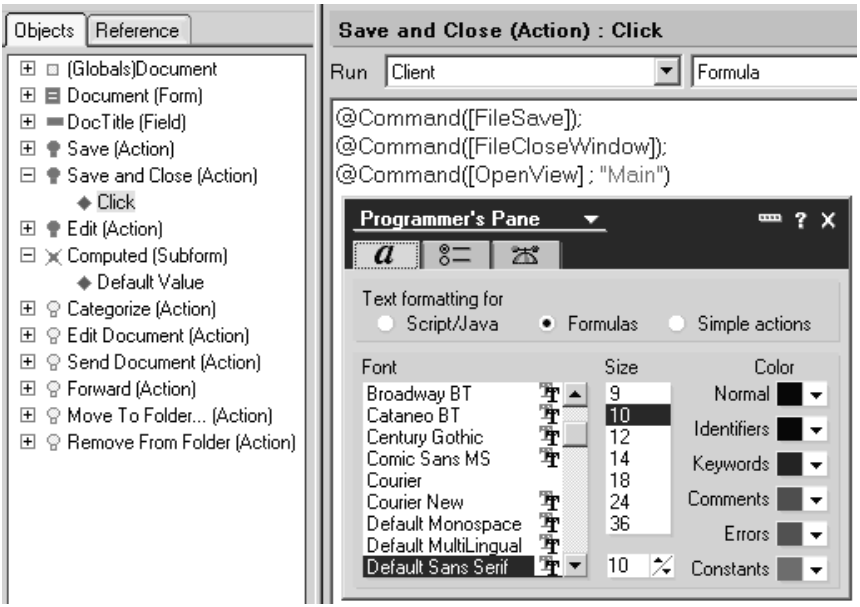
Mit jedem weiteren Buchstaben, der eingegeben wird, passt sich die Auswahl in der Liste an und zeigt nur noch die Begriffe, die den bereits geschriebenen Text enthalten. Ist der erste Teil einer Anweisung geschrieben, zeigt der Editor das Format der Anweisung als Tipp an.



Anweisungsformat

Dabei wird der Teil der Anweisung, der als nächster folgen muss, in Fettschrift angezeigt.

Der Text wird mit verschiedenen Farben dargestellt, damit die einzelnen Programmelemente deutlich zu sehen sind. Welche Farben dabei verwendet werden, bleibt dem Benutzer überlassen. Er kann die Standardfarben belassen oder eigene Farben festlegen. Die Definition der Farben wird über die Eigenschaftsbox gesteuert, die entweder über die rechte Maustaste oder über das entsprechende Symbol in der Symbolleiste angezeigt wird.

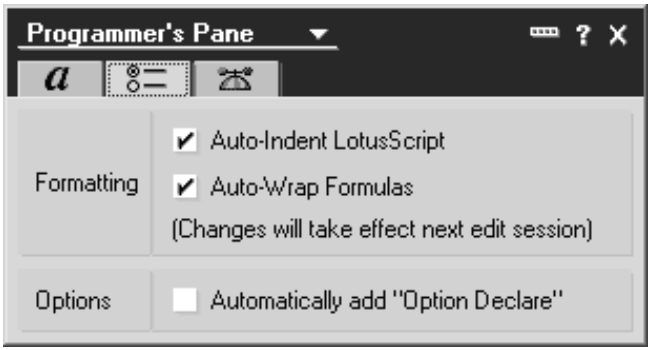


Farbschema für Programmtexte

In dem Textregister werden Schriftart und -größe festgelegt. Dazu gibt es Farben für:

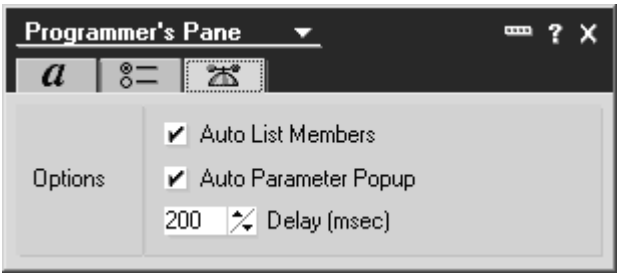
- normalen Text
- Namen
- Schlüsselwörter
- Kommentar
- Fehler
- Konstante

Im Register für Texteigenschaften wird definiert, ob in den Script-Sprachen automatisch eingerückt und bei den Formeln die Folgezeilen mit einem Standardeinzug versehen werden sollen.



Angaben zu den Texteigenschaften

Im letzten Register kann noch die automatische Anzeige der Schlüsselwortliste und der Anweisungstipps ein bzw. ausgeschaltet werden.



Angaben zur Schlüsselwortunterstützung

3

Die Datenbank

Die Grundlage aller Anwendungen unter Lotus Domino sind Datenbanken. Dabei versteht man unter ”Datenbank” eine Sammlung von Dokumenten, wobei ein Dokument die Funktion eines Datensatzes wahrnimmt. Die Dokumente sind dann unterteilt in Felder, die unterschiedliche Typen, wie z.B. Text, Zahlen oder auch Richtext enthalten können. Zudem bieten die Dokumente

weitere Gestaltungselemente wie Teilmasken, Abschnitte, Aktionen, Hotspots etc. Die Dokumentgestaltung wird im Designer als Formular grafisch realisiert und mit Programmfunktionen versehen. Die Gestaltungselemente werden als getrenntes Objekt in der Datenbank gespeichert.

Datenbanken als Container

Neben den Dokumentmasken werden auch die anderen Gestaltungsobjekte wie Ansichten, Ordner, Seiten, Rahmen etc. in den Datenbanken abgelegt. Diese Vorgehensweise verbraucht weniger Speicherplatz und ist bei einer Änderung sofort innerhalb der ganzen Datenbank verfügbar. Sichtbar werden die Änderungen oft erst, wenn die Dokumente neu gespeichert werden.

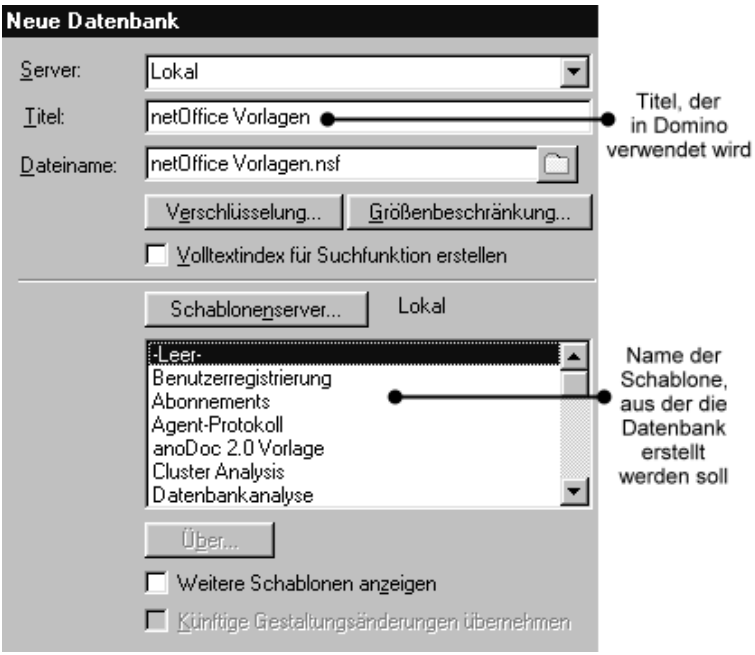
3.1 Neue Datenbank

Eine neue Datenbank kann auf unterschiedliche Arten angelegt werden:

- über eine bestehendes Template (Schablone)
- als Kopie einer bestehenden Datenbank
- durch Erstellen einer komplett neuen Datenbank.

Datenbanken können als Schablone erstellt werden. Das dient dem Ziel, die Gestaltungselemente unabhängig von echten Daten ändern zu können. Während normale Datenbanken standardmäßig mit „.nsf“ enden, erhalten Schablonen die Endung „.ntf“. Zum Erstellen einer solchen Schablonen-Datenbank verwenden Sie das Menü Datei / Datenbank / Neue Datenbank“.

Es wird das Dialogfenster „Neue Datenbank“ geöffnet.



Neue Schablonen-Datenbank anlegen

Im Titelfeld wird der Name der neuen Datenbank angegeben (z.B. netOffice-Vorlagen). In der darunter liegenden Auswahlbox, in der die vorhandenen Schablonen angeboten werden, wählt man die Option „-Leer-“, da eine neue Schablone erstellt werden soll, die von den bestehenden Schablonen unabhängig ist. Da der Designer den Datenbanknamen automatisch erstellt, ändert man die Endung ab, so dass der Datenbankname „netOffice-Vorlagen.ntf“ lautet. Mit OK wird die Schablonen-Datenbank erstellt und zu den Lesezeichen hinzugefügt.

3.1.1 Die Datenbankeigenschaften

Die oben schon erwähnten Eigenschaftsfenster sind in vielen Fällen sehr praktisch, weil alle zum gerade in Arbeit befindlichen Objekt benötigten Eigenschaften sofort im Zugriff sind. Mit dem Cursor wird die neu angelegte Datenbank in den Lesezeichen ausgewählt und dann mit der rechten Maustaste der Menüpunkt „Eigenschaften“ ausgewählt.

Datenbankeigenschaften anzeigen

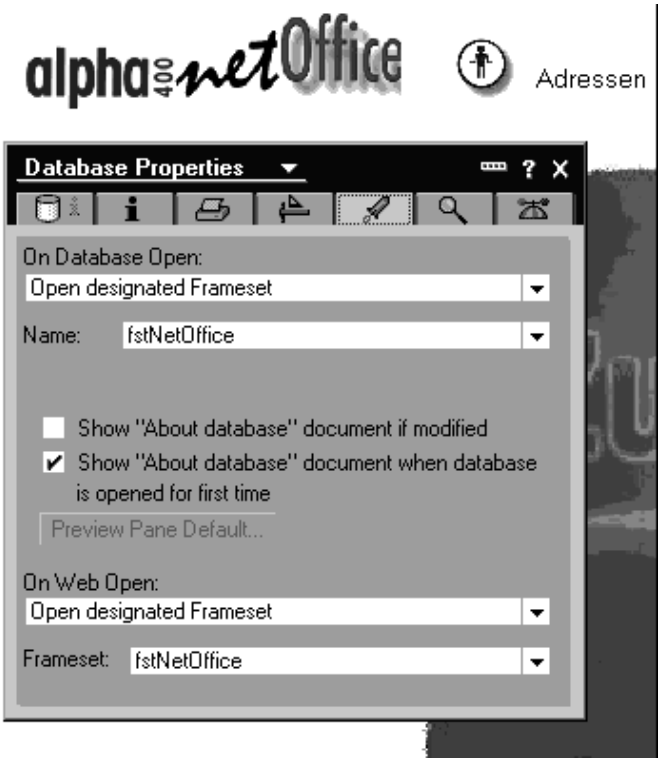
Die Eigenschaften für die Datenbank gliedern sich in mehrere Unterpunkte:

- Basisdefinitionen
- allgemeine Information
- Druckoptionen
- Gestaltungsangaben
- Startoptionen
- Indizierungseinstellungen
- weiterführende Optionen

Die Eigenschaften können zwar auch vom Benutzer bearbeitet werden, aber zum einen werden hier wichtige Grundeinstellungen vorgenommen und zum zweiten kann ein Teil der Eigenschaften vor dem Benutzer verborgen und damit seinem Einfluss entzogen werden.

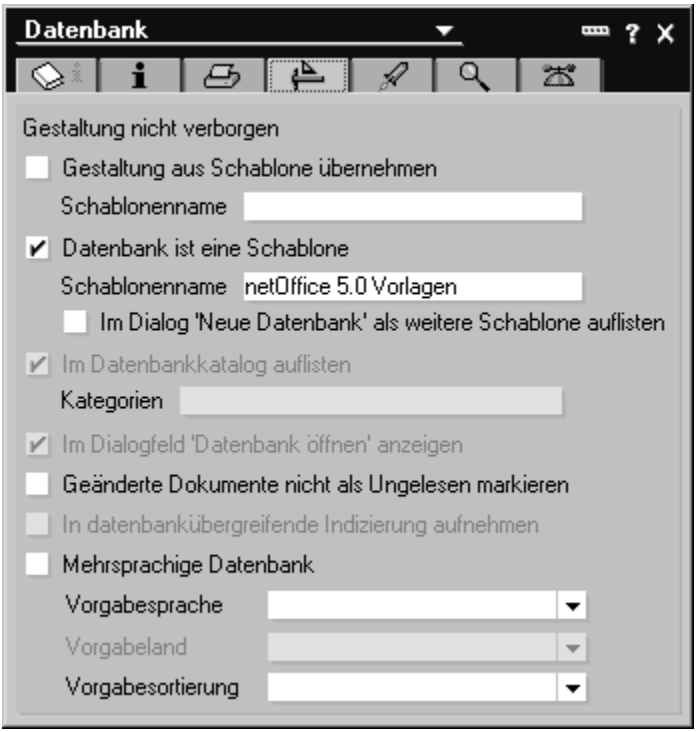
Bei den Gestaltungsoptionen ist der Name für die Schablone einzutragen. Dazu wird das Feld „Datenbank ist eine Schablone“ angehakt und das dahinter liegende Feld ausgefüllt.

Interessant sind auch die Startoptionen: Hier kann für den Datenbankstart angegeben werden, welches Gestaltungselement (Rahmengruppe, Navigator etc.) für den Anwender zuerst geöffnet werden soll.



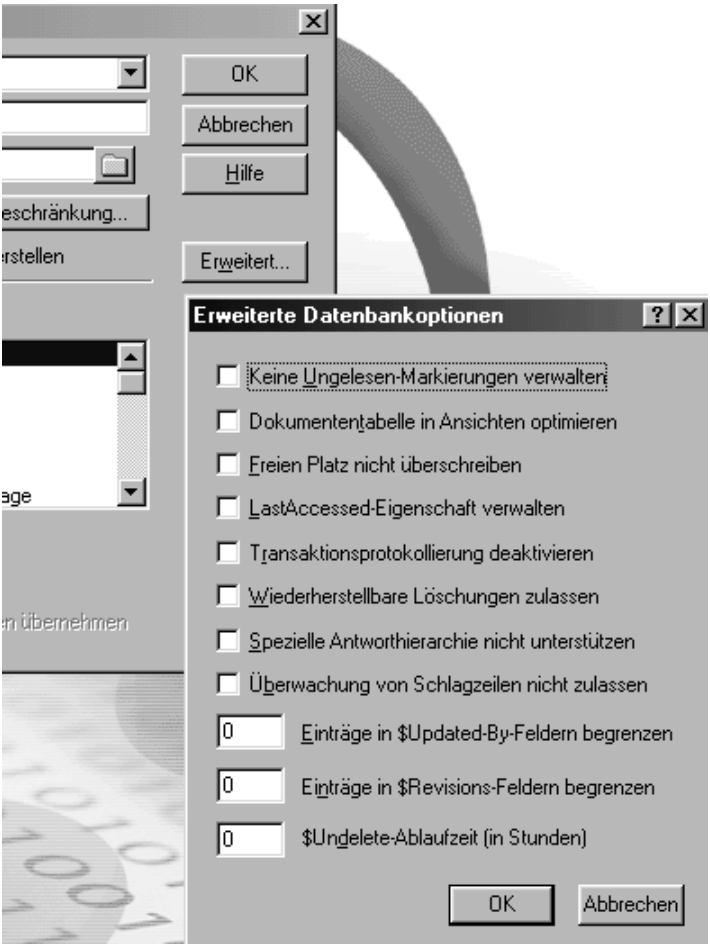
Datenbankeigenschaften – Startoptionen

Dabei kann für den Notes-Anwender ein anderer Einstieg definiert werden als für den Anwender, der über einen Browser auf die Datenbank zugreift.



Datenbankeigenschaften – Gestaltung

In den erweiterten Datenbankeigenschaften können eine ganze Reihe von Einstellungen vorgenommen werden, die im Wesentlichen zu besserer Perfomance bei der Datenbanknutzung führen.



Erweiterte Datenbankeigenschaften

Die neue Datenbank kann jetzt die einzelnen Elemente einer Domino-Anwendung aufnehmen.

Diese Seite ist zur Zeit nicht belegt.

3.1.2 ACL – Die Zugriffskontroll-Liste

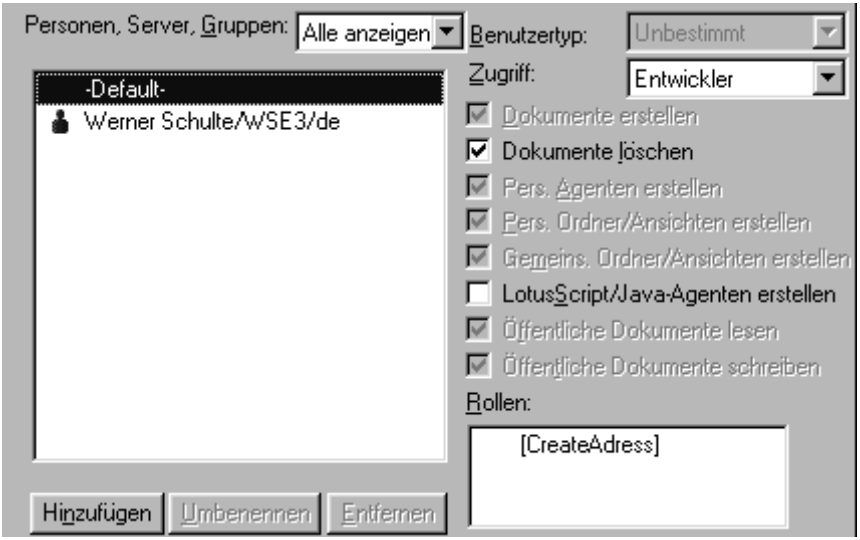
ACL bedeutet ”Access Controll List”, Zugriffsliste. Es handelt sich also um eine Aufzählung von Benutzern, die an der Datenbank bestimmte Aktionen ausführen dürfen, sowie darum, welche Rechte diese Benutzer hierbei haben.

Die Dialogbox zur Bearbeitung dieser Liste wird über das Menü ”Datei / Datenbank / Zugriff” erreicht. In der Zugriffsliste werden alle Benutzer, Benutzergruppen oder Server eingetragen, die mit der Datenbank arbeiten sollen. Die Schaltfläche ”Hinzufügen” blendet eine Auswahlbox ein, in der die Benutzer aus dem öffentlichen Namens- und Adressbuch ausgewählt werden können. Wenn mehrere Benutzer mit gleichen Rechten auf die Datenbank zugreifen sollen, ist es sinnvoll, diese Benutzer einer Benutzergruppe zuzuordnen und dann diese Benutzergruppe mit den entsprechenden Rechten in die Zugriffskontroll-Liste der Datenbank aufzunehmen.

Eine weitere Differenzierung der Benutzerrechte kann über die Vergabe von Rollen vorgenommen werden. Die Rollen können Benutzern und Benutzergruppen zugeordnet und dann gehandhabt werden wie Benutzergruppen.

Der Entwickler einer Datenbank hat z.B. andere Rechte als ein Leser oder ein Autor. Für Benutzer, die das Design der Anwendung nicht verändern sollen, ist es sinnvoll, Editoren oder Autorenrechte zu vergeben. Damit kann man neue Dokumente anlegen und bestehende Dokumente ändern.

Neben der globalen Berechtigung können einzelne Rechte festgelegt werden:



Zugriffsliste

Je nach Zugriffsrecht lassen sich die einzelnen Rechte zu- oder abschalten. Dadurch kann z.B. festgelegt werden, dass eine Gruppe Dokumente anlegen und ändern, aber nicht löschen kann.

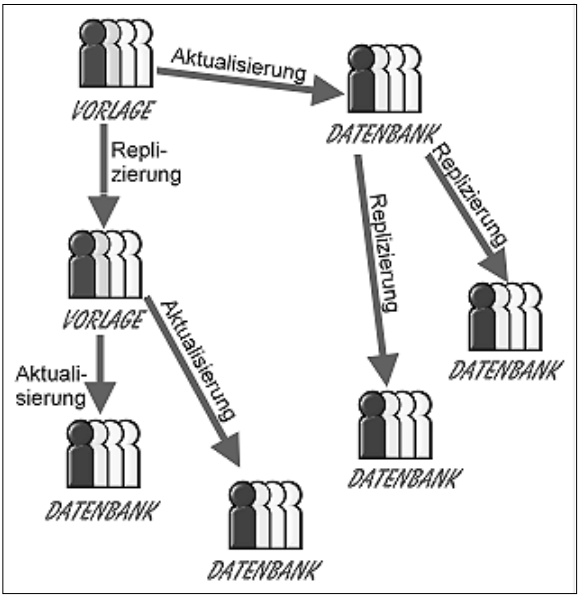
3.2 Mit Schablonen arbeiten

Die Gestaltungselemente werden unter Lotus Domino genauso als Objekte verwaltet wie die zu speichernden Datenformate und -inhalte. Daher liegt es nahe, dass die Anwendung zusammen mit den Daten in Datenbanken gespeichert wird. Diese Vorgehensweise bietet erhebliche Vorteile:

- Die Verteilung von geänderter Software kann ablaufen wie diejenige von Daten (nämlich durch Replikation).
- Die Anzahl der zu verteilenden Objekte beschränkt sich auf die Datenbanken bzw. deren Vorlagen.
- Die Speicherung der Anwendungsdaten erfolgt strukturiert und kann direkt zum Test der Entwicklungselemente verwendet werden. Test und Programmierung sind im unmittelbaren Zusammenhang durchzuführen.
- Der Zugriff auf Daten und Gestaltung erfolgt über eine einheitliche Oberfläche.

Eine wesentliche Erleichterung für die Anwendungsentwicklung stellt die Arbeit mit Datenbankvorlagen oder -schablonen dar. Gekennzeichnet durch die Dateierweiterung „.ntf“ werden Datenbanken ausschließlich für die Aufnahme der Gestaltungselemente reserviert. Die Datenbanken zur Datenhaltung erhalten einen Verweis auf die zuständige Vorlage. Dadurch wird zwar ein Teil der oben genannten Vorteile wieder aufgegeben, es wird aber auf der anderen Seite die Möglichkeit gewonnen, Datenbanken unabhängig von den darin enthaltenen Daten zu aktualisieren.

Datenbankvorlagen werden auf einem zentralen Server abgelegt und notwendige Änderungen vorgenommen. Erst wenn die Gestaltungsänderung abgeschlossen ist, wird die Schablone auf dem Server für die aktiven Vorlagen platziert.



Datenbankvorlagen

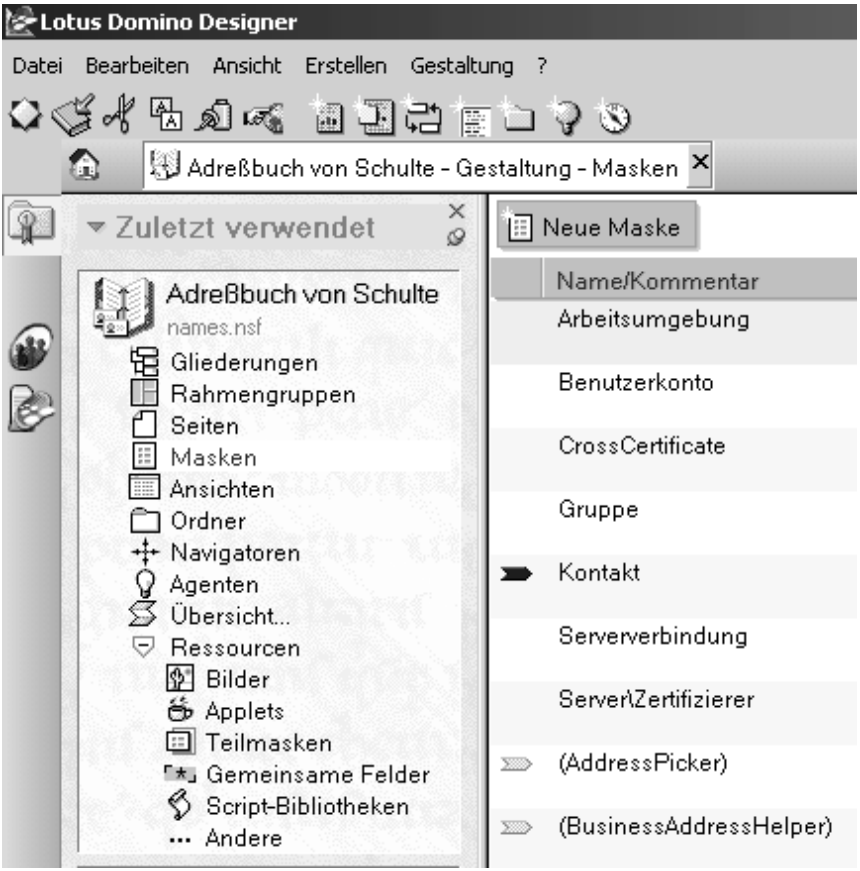
Mit Hilfe eines Server-Prozesses (Design) werden automatisch alle Vorlagenänderungen von hier aus auf die entsprechenden Datenbanken übertragen. Zusammen mit der Replikationsfunktion lassen sich Domino-Datenbanken, die über viele Server und Clients verteilt sind, schnell und ohne großen Aufwand aktualisieren.

Es ist möglich, Repliken der Vorlagen zu verwenden, um die Arbeit der Aktualisierung auf mehrere Server zu verteilen. Es können aber auch alle Datenbanken zentral aktualisiert und dann repliziert werden.

4

Gestaltungselemente

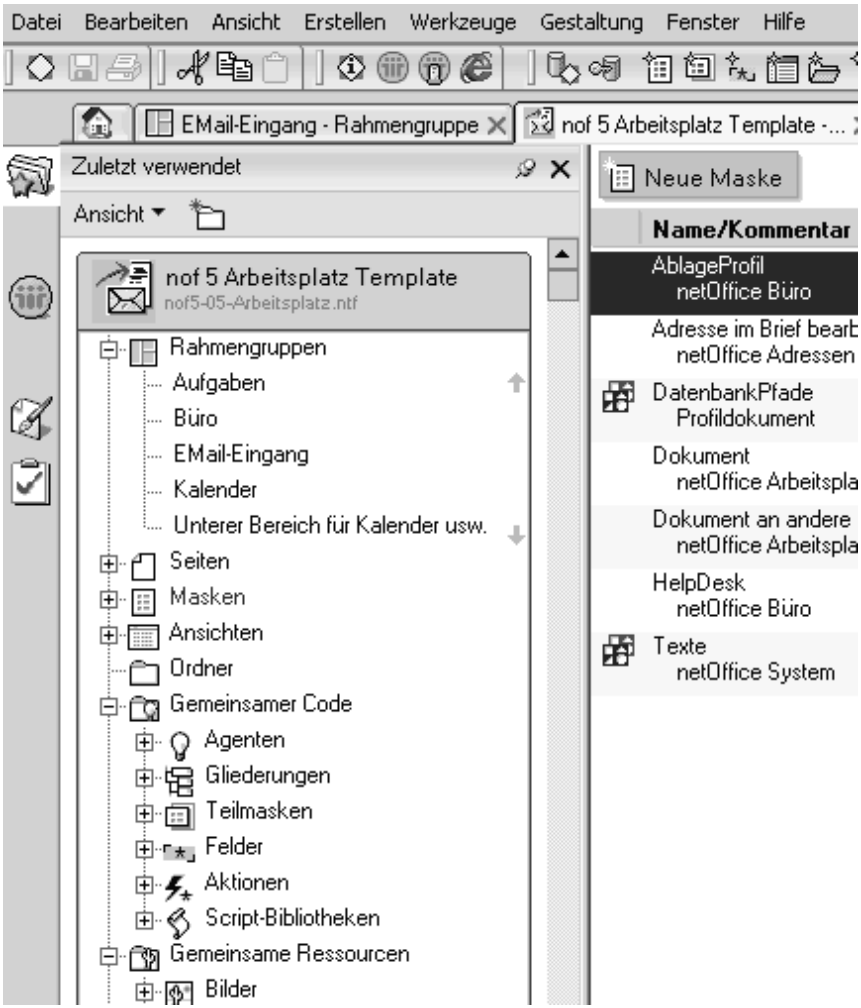
Für jede Gestaltungsart passt sich die Entwicklungsumgebung entsprechend an. In Release 5 wird bereits eine breite Palette von Gestaltungselementen angeboten. Neu sind dabei vor allem Elemente, die zur Entwicklung von Internetseiten erforderlich sind – also z.B.: Seiten, Rahmen und Rahmengruppen, Bilder und Applets.



Gestaltungselemente bei R5

In Release 6 werden neue Elemente eingeführt und die bestehenden noch benutzerfreundlicher gestaltet. Während in Release 5 auf der linken Seite der Benutzeroberfläche lediglich die Elementgruppen aufgeführt sind und die einzelnen Elemente nur auf der rechten Seite aufgelistet werden, sind unter Release 6 auch ein-

zelne Elemente in Teillisten unter den Elementgruppen der linken Seite zu sehen und damit auch direkt anzeigbar.



Gestaltungselemente bei R6

Auch die Unterteilung der Elemente wurde neu vorgenommen. Die „Ressourcen“ wurden unterteilt in „Gemeinsamer Code“ und „Gemeinsame Ressourcen“. Und die Gruppe „Andere ...“ wurde ausgebaut. Unter der Gruppe „Gemeinsamer Code“ wurden die Gestaltungselemente zusammengefasst für:

- Agenten
- Gliederungen
- Teilmasken
- Felder
- Aktionen
- Script-Bibliotheken

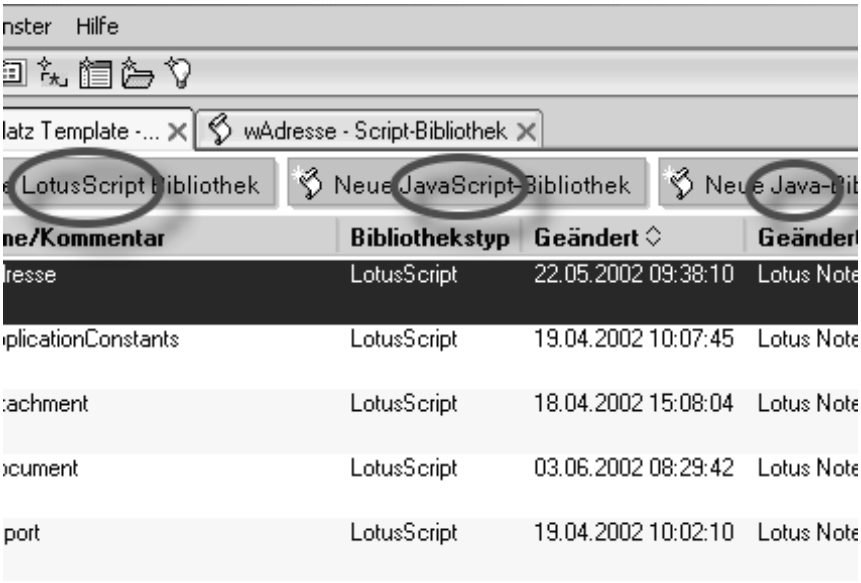
Die Gruppe „Gemeinsame Ressourcen“ enthält die Gestaltungselemente für:

- Bilder
- Dateien
- Applets
- Stylesheets
- Datenverbindungen

In der Gruppe „Andere …“ sind zusammengefasst:

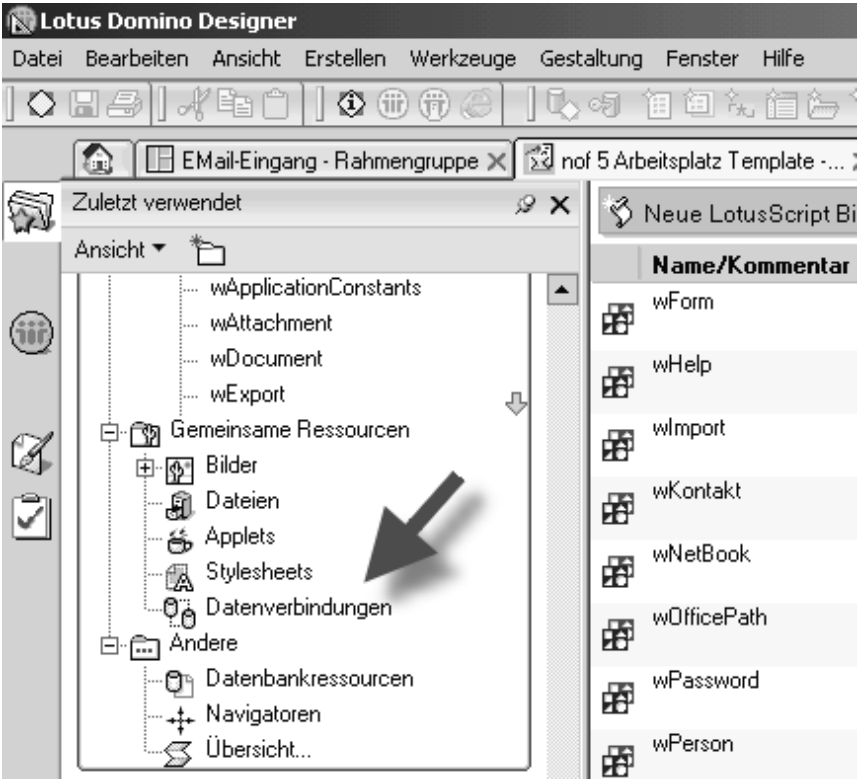
- Datenbankressourcen
- Navigatoren
- Synopse

Bei den Script-Bibliotheken können bei Release 5 Programmsammlungen für LotusScript und Java angelegt werden. In Release 6 ist es auch möglich, Script-Bibliotheken für Javascript zu erstellen.



Script-Bibliotheken bei R6

Eine zusätzliche Verbesserung stellt die Einbeziehung von Stylesheets und Datenverbindungen dar, die bei Release 6 vorgenommen wurden.

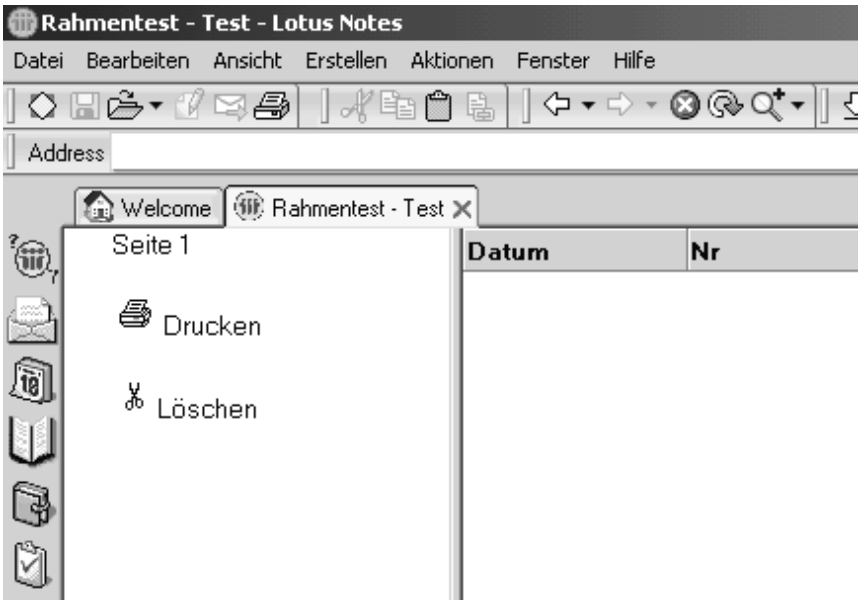


Stylesheets und Datenverbindungen bei R6

Die Unterstützung des Entwicklers wurde in den Releases 5 und 6 stark ausgebaut. Sowohl für die Entwicklung von Notes-Anwendungen als auch für die von Internet-Lösungen wurden vielseitige Werkzeuge in übersichtlicher Form zur Verfügung gestellt.

4.1 Rahmen und Rahmengruppen

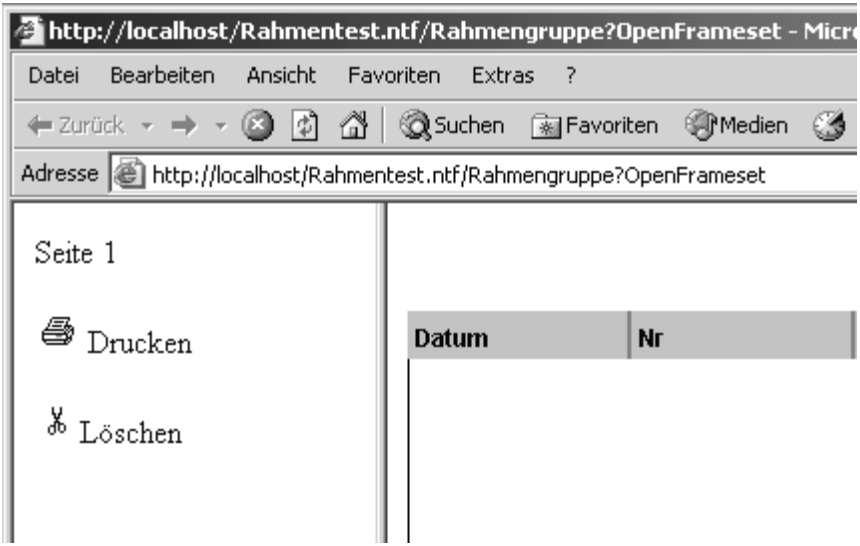
Das aus dem Internet bekannte Konzept der Rahmen und Rahmengruppen wurde mit Release 5 voll in die Designerwerkzeuge integriert. Dadurch wird ein einheitliches Layout für den Notes-Client und Web-Browser ermöglicht.



Einfache Rahmengruppe

Die Anzeige kann für die beiden Anwendungsformen in einem Zuge bewerkstelligt werden. Der Benutzer eines Notes-Clients greift auf das gleiche Element zu, das auch der Browser-Benutzer verwendet. Je nach Einstellung der Elementeigenschaften und den Möglichkeiten des verwendeten Browsers ist die Darstellung

in den Benutzeroberflächen ähnlich oder nur in den Funktionen gleich. Insbesondere die zur Verwendung von IBM angebotenen Java-Applets zur Anzeige bestimmter Gestaltungselemente (wie z.B. Ansichten) ermöglichen eine gute Angleichung der Gestaltung in unterschiedlichen Client-Programmen.



Rahmengruppe im Browser

Eine Rahmengruppe dient ausschließlich dem Zweck, Inhalte zu verwalten und darzustellen. Sie stellt einen Fensterinhalt in getrennten Bereichen, den Rahmen, dar. Der Entwickler kann bei den verschiedenen Aktionen oder Gestaltungselementen steuern, in welchem Rahmen die Auswirkung eines Events ablaufen soll. Diese Technik erlaubt es, Inhalte innerhalb eines Fensters genau zu positionieren, so wie es auch mithilfe von Tabellen durchzu-

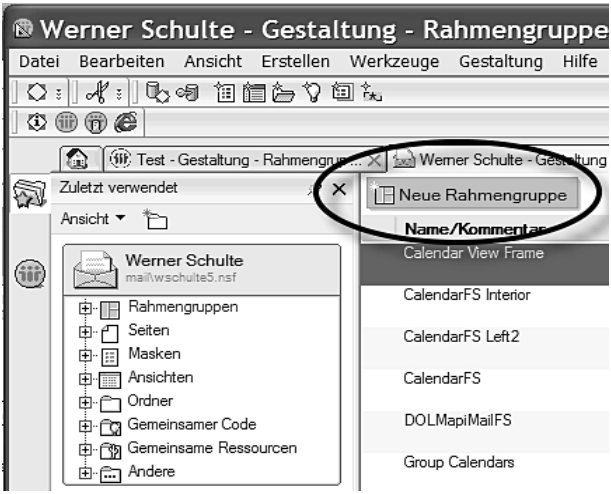
führen wäre. Der spezielle Effekt der Rahmen besteht jedoch darin, dass der Inhalt eines Rahmens ausgetauscht werden kann, ohne den gesamten Bildschirminhalt neu zu senden. Die Einbindung der Rahmentechnik in die Gestaltungswerkzeuge des Designers und deren Verwendung bei der Darstellung von Inhalten im Notes-Client stellt ein Beispiel dafür dar, wie der Notes-Client von Internet-Techniken profitieren und die Entwicklung von Internet-Anwendungen mit Notes verbessert werden kann. Leider hat IBM die Entwicklung in diese Richtung nicht bei allen Gestaltungselementen ebenso konsequent weitergeführt.

Diese Seite ist zur Zeit nicht belegt.

4.1.1 Rahmengruppe und Rahmen erstellen

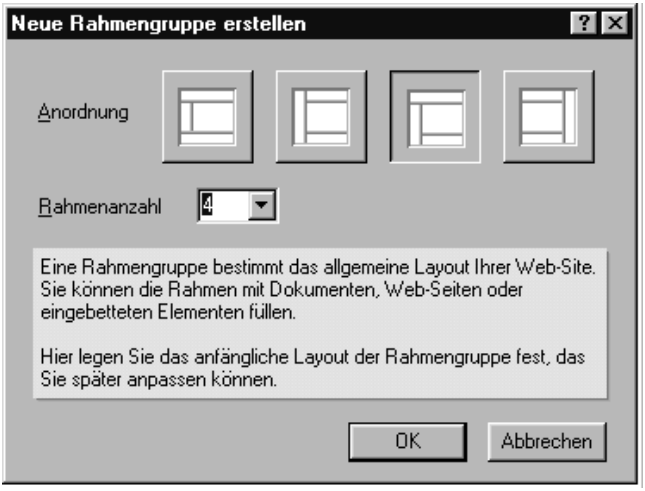
Rahmengruppen dienen dazu, Inhalte attraktiv und flexibel in einem Fenster darzustellen. Das Fenster wird dabei in verschiedene Bereiche aufgeteilt und jeder Bereich kann eigene Gestaltungselemente wie Seiten, Masken, Ansichten oder auch URLs zu Internetseiten aufnehmen.

Wie viele Rahmen man wählt und wie dabei die Aufteilung des Fensters erfolgt, hängt von den Anforderungen der aktuellen Problemstellung, aber auch von generellen Gestaltungsrichtlinien ab. Bevor eine Anwendung mit Rahmengruppen erstellt wird, ist es deshalb sehr wichtig, eine genaue Vorstellung von dem Aussehen und der Funktion der Rahmeninhalte zu haben. Am besten versucht man zunächst eine Zeichnung oder Grafik für das Aussehen des Fensters anzufertigen. Zur Erstellung einer Rahmengruppe verwendet man das Menü „Erstellen / Gestaltung / Rahmengruppe“ oder die Schaltfläche „Neue Rahmengruppe“.



Schaltfläche „Neue Rahmengruppe“

In der folgenden Dialogbox wird nun angegeben, wie viele Rahmen erstellt werden und wie die Rahmen angeordnet werden sollen.

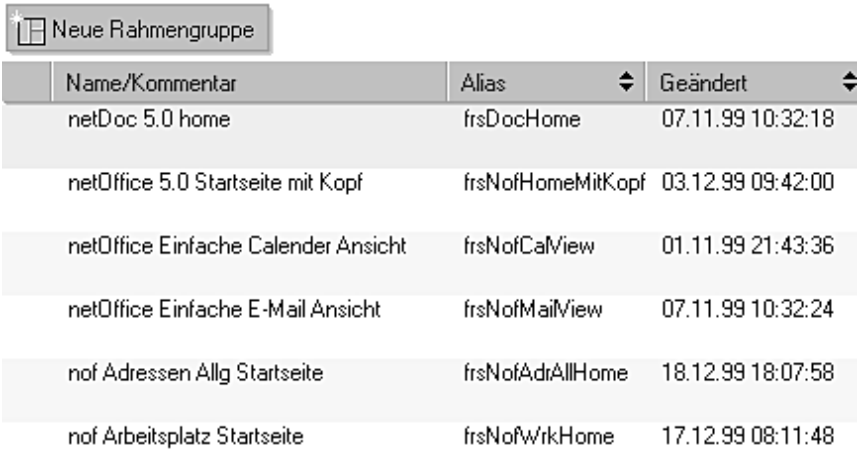


Neue Rahmengruppe erstellen

In dem Feld Rahmenanzahl wird als höchster Wert eine vier vorgegeben. Wenn mehr Rahmen gewünscht werden, können in der nun folgenden Ansicht auch noch weitere Rahmen durch Klicken auf die Schaltfläche „In Zeilen aufteilen“ hinzugefügt werden. Der Designer erstellt jetzt eine leere Rahmengruppe, mit der noch nicht viel anzufangen ist. In den einzelnen Rahmen können unterschiedliche Gestaltungsobjekte angezeigt werden: Seiten, Masken, Ordner, Ansichten oder Navigatoren. Es ist auch möglich, Rahmengruppen zu verschachteln, also eine neue Rahmengruppe in einem Rahmen zu zeigen. Bevor weitere Entwicklungsschritte vorgenommen werden, sollte die Rahmengruppe gespeichert werden. Es ist empfehlenswert, der Rahmengruppe neben dem Namen auch ein Alias für die Verwendung in Programmen zu geben.

4.1.2 Mit Rahmengruppen arbeiten

Bestehende Rahmengruppen werden in der rechten Seite des Designers aufgelistet und durch Doppelklicken geöffnet. Zeigt die Auflistung keine Rahmengruppen, sondern andere Gestaltungselemente, so kann man diese Anzeige ändern, indem man links in der Lesezeichenseite der Datenbank zu dem Punkt „Rahmengruppen“ wechselt.



Auflistung der Rahmengruppen

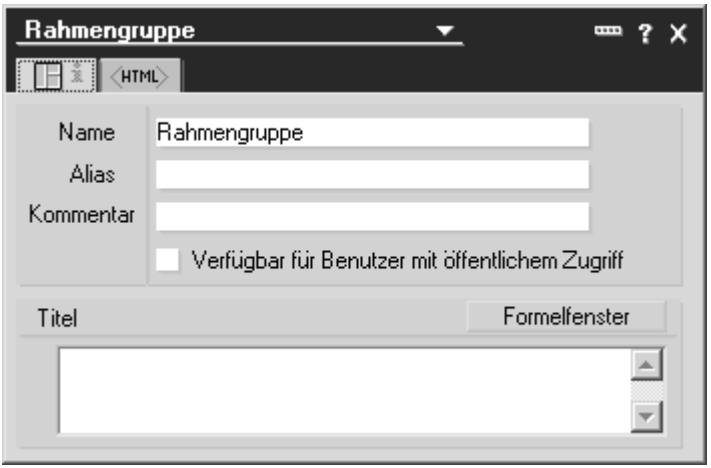
Die eigentlichen Steuerungsaufgaben übernehmen die Rahmen innerhalb der Rahmengruppe. Deshalb verfügt die Rahmengruppe selbst nur über einige wenige Eigenschaften. Sie werden in dem Eigenschaftsdialog bearbeitet, der über das entsprechende Symbol geöffnet wird.

Der Dialog enthält zwei Register:

- Allgemeine Eigenschaften
- HTML

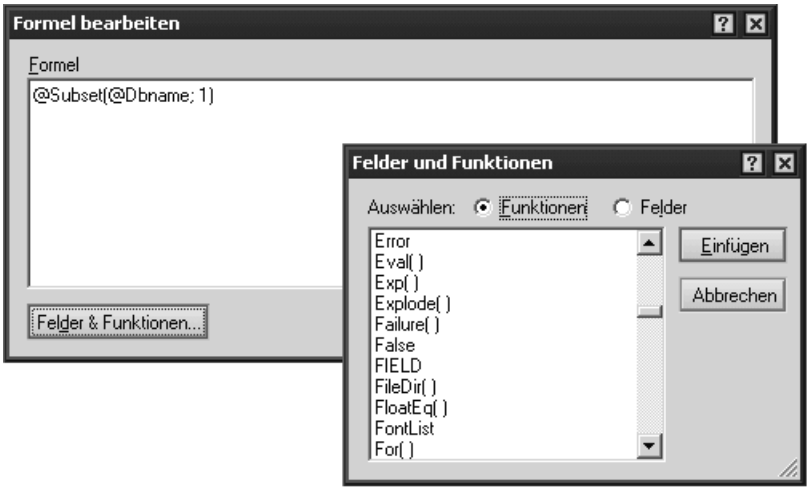
Der Inhalt des HTML-Registers wird in der Regel nicht benötigt, da diese Funktionen bei den Rahmen hinterlegt sind.

In dem Register „Allgemein“ ist die Formel zur Ermittlung des Titels für die Rahmengruppe interessant.



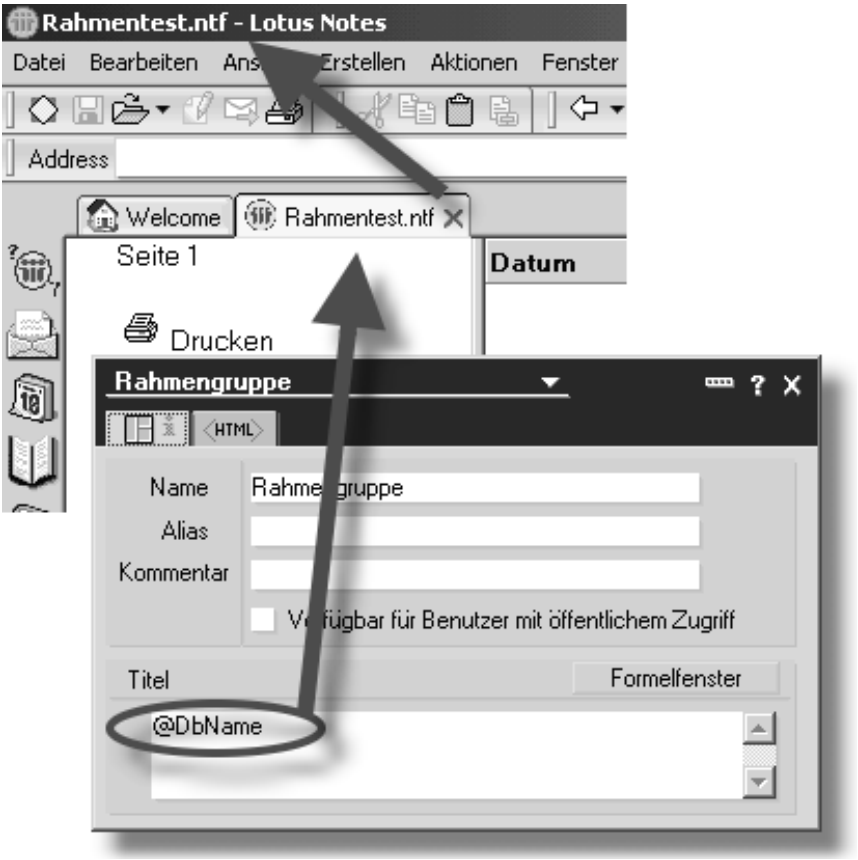
Allgemeine Eigenschaften der Rahmengruppe

In diesen Formelbereich kann ein Begriff oder eine Formel eingetragen werden. Die Formel muss als Ergebnis einen Begriff bzw. einen Text ergeben, der im Register des Notes-Fensters erscheint, wenn die Rahmengruppe angezeigt wird. Zur Erfassung der Formel kann ein Formelfenster geöffnet werden, das die Eingabe der Formel unterstützt.



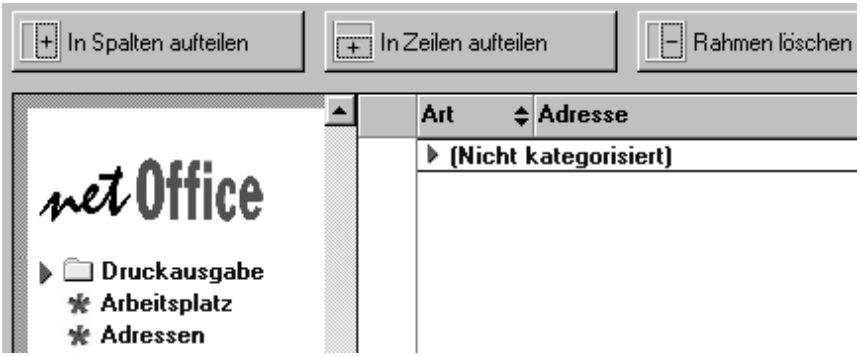
Formel bearbeiten

In der Anzeige der Rahmengruppe in dem Notes-Client wird die Formel umgesetzt und in das Windowsfenster übernommen, wenn das entsprechende Notes-Fenster ausgewählt ist.



Formel und Fenstertitel

Die Rahmengruppe kann jetzt in einzelne Rahmen aufgeteilt werden. Am oberen Rand des Fensters wird dazu eine Leiste mit Schaltflächen gezeigt, die Aufteilungsfunktionen zur Verfügung stellen. Soll ein Rahmen weiter unterteilt werden, so wird er ausgewählt und über die Schaltflächen „In Spalten aufteilen" bzw. „In Zeilen aufteilen" oder mit den entsprechenden Löschtasten angepasst.



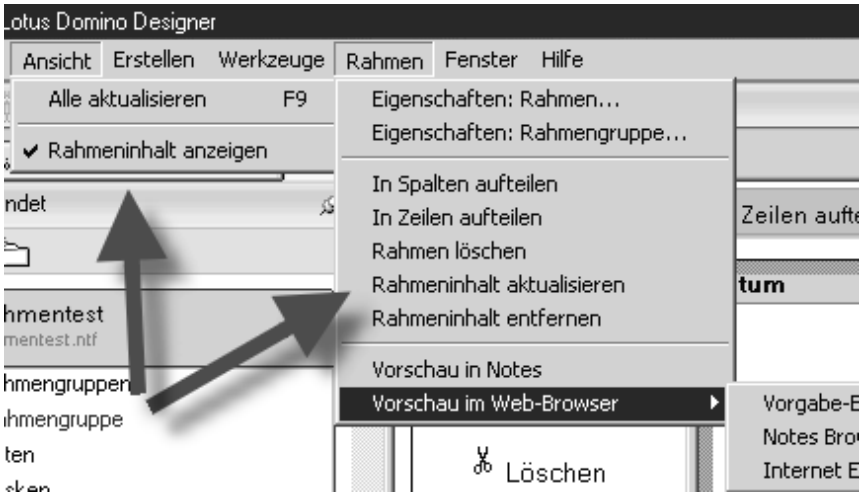
Schaltflächen zur Anpassung der Rahmengruppe

Soll z.B. ein Logo im linken Rahmen unabhängig von dem restlichen Rahmen darunter sein, so unterteilt man den ganz linken Rahmen noch einmal mit Hilfe der Schaltfläche „In Zeilen aufteilen". Folgende Schaltflächen können genutzt werden:

- Rahmengruppe speichern (Release 6)
- In Spalten aufteilen
- In Zeilen aufteilen
- Rahmen löschen

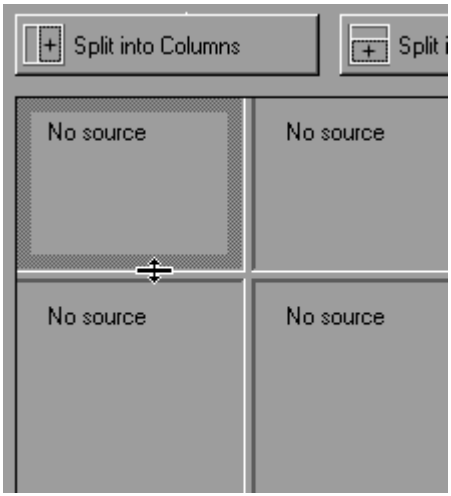
- Rahmeninhalt entfernen
- Horizontal spiegeln (Release 6)

Neben den Schaltflächen können auch die Menüpositionen des Windows-Menüs verwendet werden. Das Menü „Rahmen“ enthält eine zusätzliche Position: „Rahmeninhalt aktualisieren“. Im Menü „Ansicht“ kann eine Aktualisierung aller Rauminhalte ausgelöst werden. Interessant ist in diesem Menü auch die Position „Rahmeninhalt anzeigen“, die standardmäßig aktiviert ist. Wird die Position angeklickt, ändert sich die Anzeige des Rahmeninhalts aller Rahmen der Rahmengruppe. Anstelle der Inhalte wird lediglich deren Beschreibung angezeigt. Das ist dann wichtig, wenn in den Rahmen Web-Seiten angezeigt werden und bei der Entwicklung nicht ständig auf das Internet zugegriffen werden soll.



Menü für Rahmen und Rahmengruppen

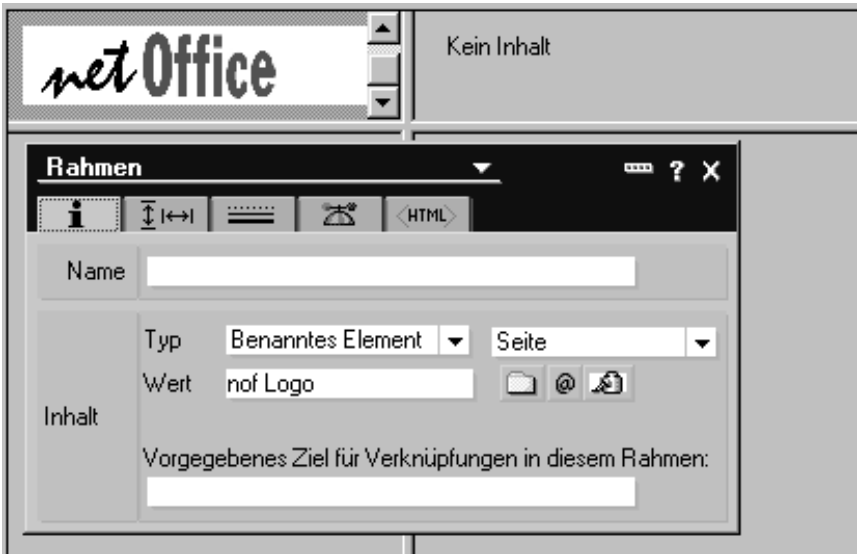
Höhe und Breite der einzelnen Rahmen werden durch Anklicken mit der linken Maustaste und Verschieben in die gewünschte Position angepasst.



Rahmen anpassen

Das weitere Vorgehen hängt von dem Element ab, das in einen Rahmen eingefügt werden soll. Soll also ein Logo in einen Rahmen eingefügt werden, so ist zunächst eine Seite anzulegen und das Logo als Grafik einzufügen (siehe Kapitel „Seiten und Masken“).

Über die Eigenschaften des Rahmens wird die Seite mit dem Logo dem Rahmen zugeordnet.



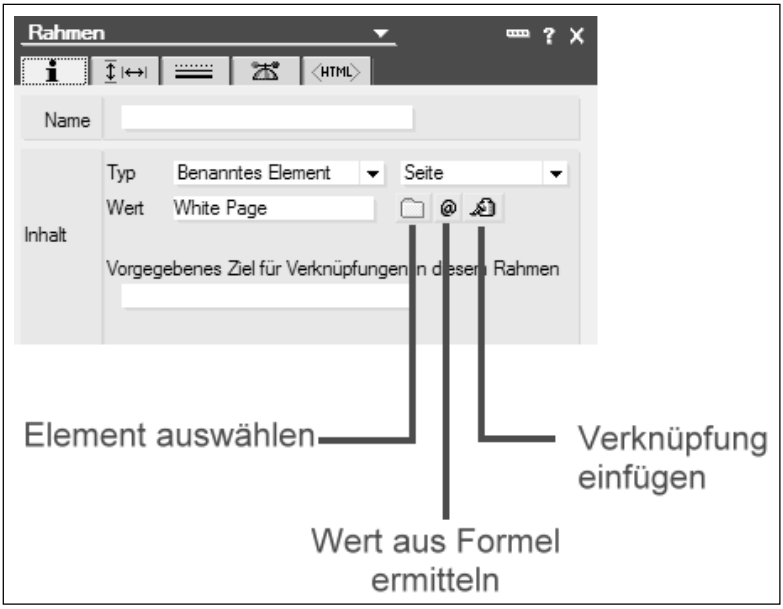
Eigenschaften „Rahmen“

Der Name für den Rahmen wird nur dann benötigt, wenn er bei der Ausgabe eines anderen Rahmens verwendet werden soll. Der Inhalt des Rahmens wird sofort angezeigt, wenn das ausgewählte Element zur Entwicklungszeit zur Verfügung steht. Ist das nicht der Fall, kommt es zur Anzeige einer Fehlermeldung in dem Rahmen. Der Fehler kann übergangen werden, wenn der Inhalt zur Laufzeit der Anwendung verfügbar ist.

Als Typ können angegeben werden:

- **URL**
  Adresse einer Webseite (z.B. http://www.midrangemagazin.de)
- **Benanntes Element**
  Gestaltungselement (z.B. Seite, Maske, Ansicht usw.)
- **Verknüpfung**
  Notes-Verknüpfung (z.B. zu einem Dokument)

Die Festlegung des eigentlichen Elements kann dann über das Einfügen der Zwischenablage, eine Formel oder mit Hilfe des Dialoges „Objekt einfügen“ erfolgen. Dabei hängt es vom Typ ab, welche dieser Einfügeversionen angeboten werden.

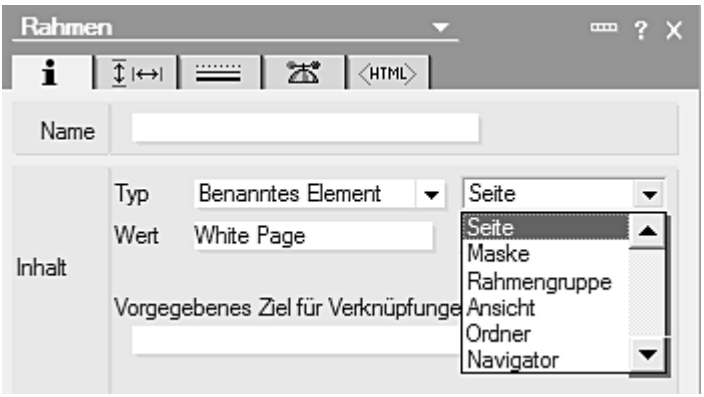


Wert für Rahmeninhalt festlegen

Soll ein Gestaltungselement in den Rahmen eingefügt werden, so steht dafür das Auswahlfeld auf der rechten Seite des Eigenschaftsfensters bereit. Der Wert „Seite“ wird angezeigt. Insgesamt können folgende Elemente eingesetzt werden:

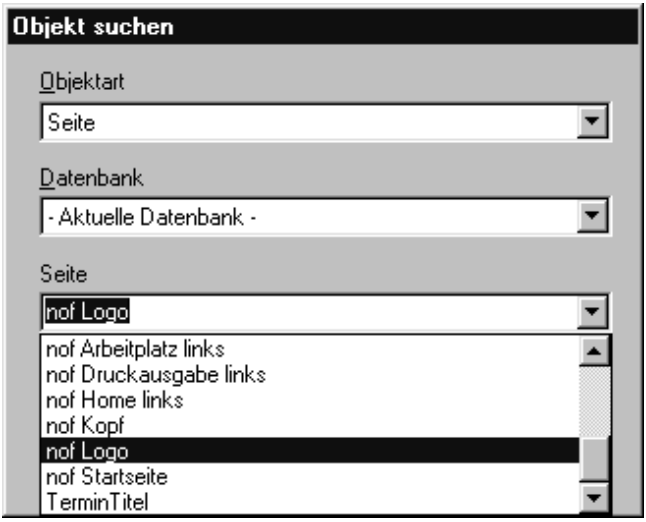
- Seite
- Maske
- Rahmengruppe
- Ansicht
- Ordner
- Navigator

Im Feld „Wert“ kann dann der Name des gewünschten Elements eingetragen werden.



Element auswählen

Handelt es sich bei dem Typ um ein „Benanntes Element“, so kann über die daneben positionierte Auswahlschaltfläche ein Dialogfenster geöffnet werden, in dem alle Angaben eines Elements zum Eintragen bzw. zum Auswählen bereitstehen. In diesem Dialog kann auch die Datenbank und die Elementart gewechselt werden.

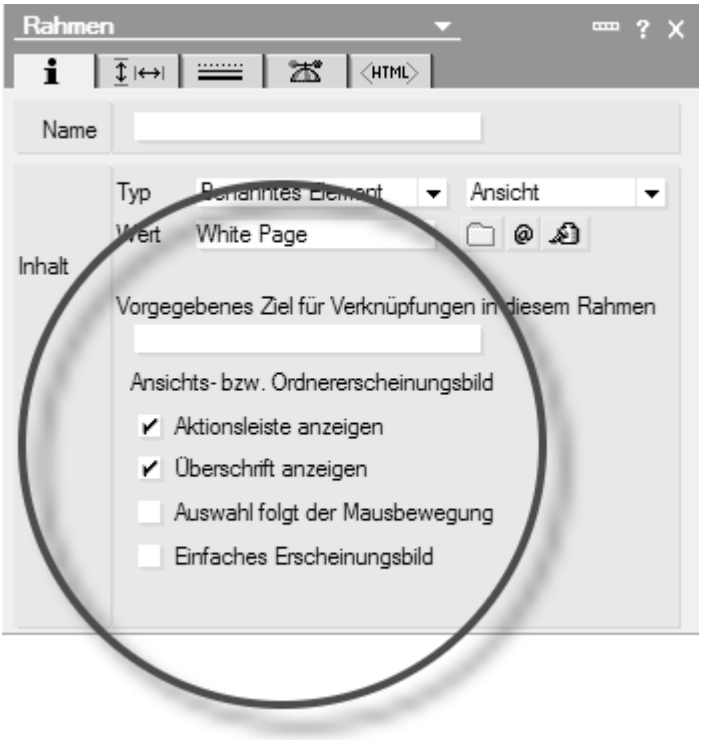


Seite einfügen

Auf diesem Weg können damit auch ohne Probleme Standarddatenbanken wie die Maildatenbank oder das Adressbuch des Benutzers mit ihren Ansichten in einem solchen Rahmen gezeigt werden. Wird eine Ansicht als Element eingesetzt, so erscheinen zusätzliche Felder in der Eigenschaftsbox des Rahmens. Sie definieren das äußere Erscheinungsbild der Ansicht im Rahmen:

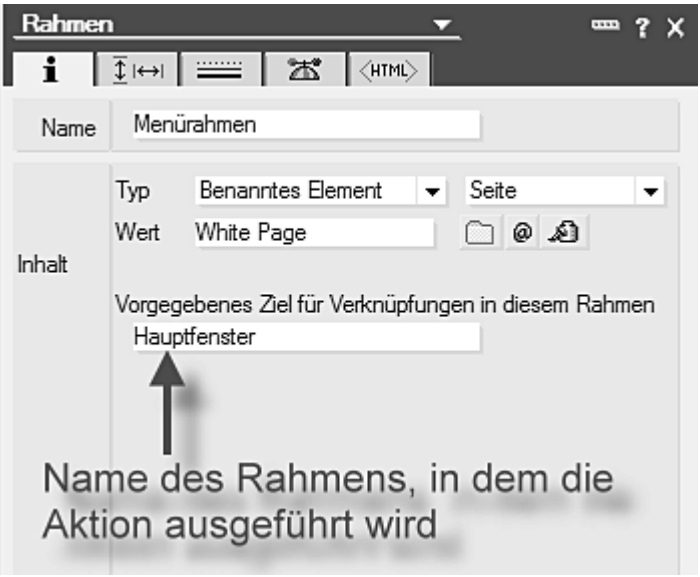
- Sollen die Aktionsschaltflächen der Ansicht angezeigt werden?
- Sollen die Überschriften am Kopf der Spalten der Ansicht gezeigt werden?

- Soll das Dokument ausgewählt werden, wenn der Cursor darauf zeigt?
- Soll nur die Ansicht (ohne weitere Elemente der Ausgangsdatenbank) angezeigt werden?



Anzeigeoptionen für Ansichten

Enthält ein Gestaltungselement innerhalb eines Rahmens eine oder mehrere Aktionen, so wird bei den Rahmeneigenschaften festgelegt, in welchen Rahmen die Ergebnisse der Aktionen ausgegeben werden sollen. Dieser Eintrag gilt pauschal für alle Aktionen innerhalb dieses Rahmens.



Zielrahmen festlegen

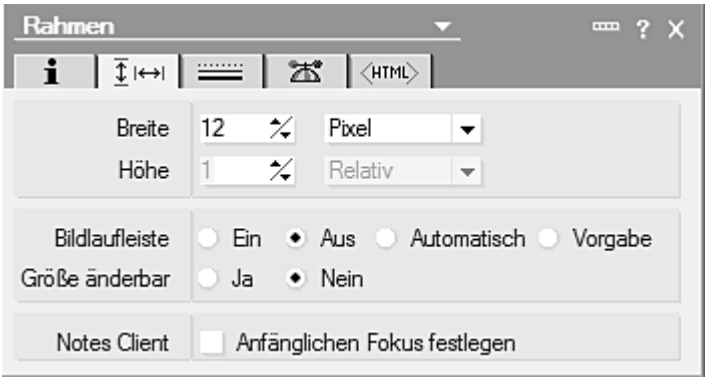
Der Zielrahmen kann auch bei anderen Gestaltungselementen eingetragen werden. Ist das der Fall, so werden die Angaben der jeweils untersten Ebene verwendet.

Hinter den Reitern im Eigenschaftsfenster verbergen sich noch weitere interessante Eigenschaften des Rahmens. Bei dem zweiten Reiter „Ränder“ kann die Bildlaufleiste abgeschaltet werden.

Eigenschaften „Rahmenränder“

Das ist bei den Rahmen mit festem Inhalt, der in den Rahmen passt, sinnvoll. Genauso kann man hier an- und abschalten, ob die Größe des Rahmens vom Anwender zur Laufzeit verändert werden darf. Leider kann man diese Eigenschaft nur für den kompletten Rahmen einstellen.

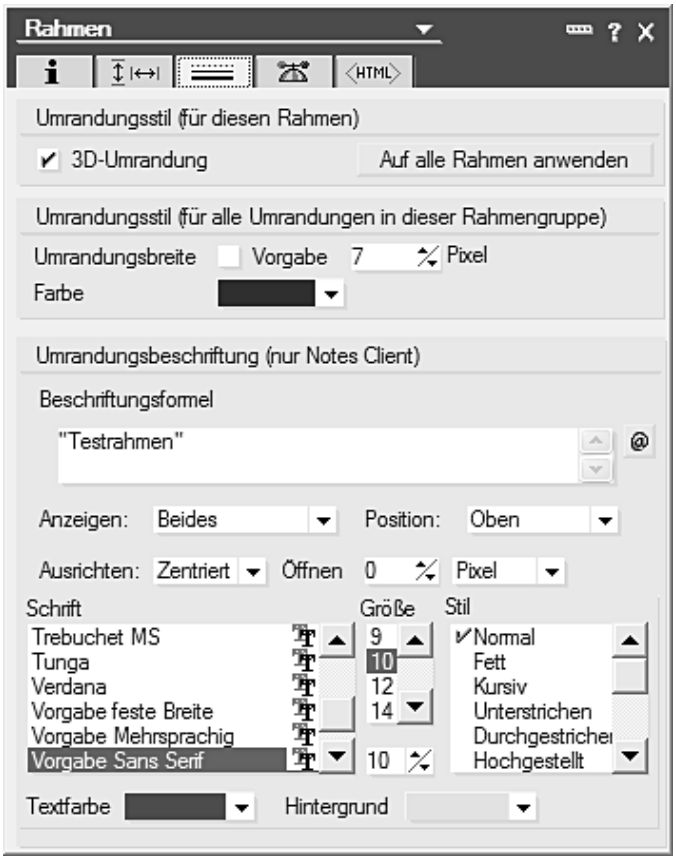
In Release 6 wurde zusätzlich eingeführt, dass festgelegt werden kann, ob der Rahmen beim Öffnen der Rahmengruppe als Erstes den Cursor erhält.



Fokus festlegen in Release 6

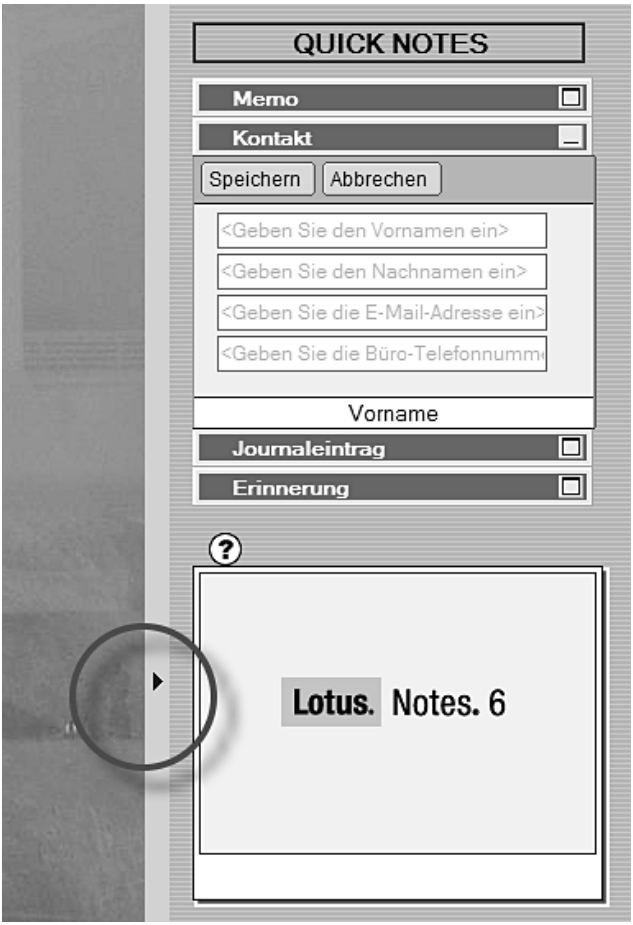
Hinter dem dritten Reiter „Rahmen“ verbirgt sich die Einstellung für die Rahmenbreite.Damit sie verstellt werden kann, muss der Haken „Umrandungsbreite“ deaktiviert werden. Dann kann die Breite in Pixeln eingestellt werden. Allerdings muss die Breite mindestens 5 Pixel betragen, sonst wird nichts dargestellt.

Die Angaben unter diesem Register werden in Release 6 um eine neue Möglichkeit erweitert. Der Entwickler kann hier angeben, ob der Rahmen ein- bzw. ausgeblendet werden soll. Diese Rahmen werden dann angezeigt bzw. verborgen, wenn man ein kleines Dreieck am Rand des Rahmens anklickt. In der Maildatenbank von Release 6 wurde diese neue Rahmeneigenschaft intensiv genutzt. Sowohl die Dokumentenvorschau als auch die Verwendung der „Quick-Notes“ wurden auf diesem Wege erstellt.



Rahmenumrandung in Release 6

Die Beschriftung kann als Zeichenkette oder über eine Formel erzeugt werden. Nur wenn in dem Feld „Anzeigen“ ein Wert eingestellt wird, der von „Keines“ abweicht, wird der Rahmen mit dieser Eigenschaft ausgestattet. Die Beschriftung wird nur dann angezeigt, wenn sie am oberen bzw. unteren Rand des Rahmens positioniert wird.

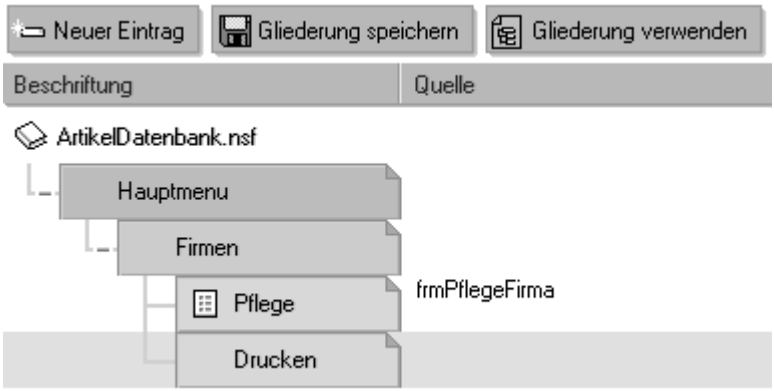


Aufgeklappter Rahmen in der Maildatenbank

Über die Menüpunkte „Rahmen / Vorschau im Web Browser“, „Rahmen / Vorschau in Notes“ oder über die entsprechenden Icons ganz rechts oben kann die Rahmengruppe so angezeigt werden, wie sie später die Benutzer sehen. Dabei wird zunächst die Rahmengruppe gespeichert und dann der Notes-Client bzw. der angegebene Browser zur Anzeige gestartet.

4.2 Gliederungen

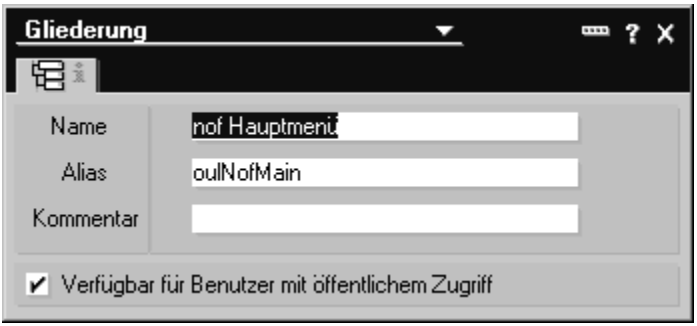
Zur Steuerung der Rahmeninhalte stellt der Designer neben Navigatoren und Imagemaps das Gestaltungselement „Gliederung“ zur Verfügung. Die Gliederung kann flach (ohne aufklappbare Gliederungspunkte) sein oder eine Baumstruktur erhalten. Für die Erstellung von Gliederungen wird eine eigene Oberfläche verwendet, die es erlaubt, die Struktur der Gliederung zu sehen und die einzelnen Einträge gezielt zu bearbeiten.



Gliederung bearbeiten

Ein neuer Gliederungspunkt wird über das Menü oder die Schaltfläche „Neuer Eintrag“ an der Cursorposition erzeugt. Über die beiden Schaltflächen „Eintrag einrücken“ bzw. „Eintrag ausrükken“ werden die Punkte in der Gliederungshierarchie zu Unterpunkten oder zu Unterpunkten von Unterpunkten.

Die Gliederung kennt nur wenige Eigenschaften: Name, Alias und „Verfügbar für Benutzer mit öffentlichem Zugriff“.

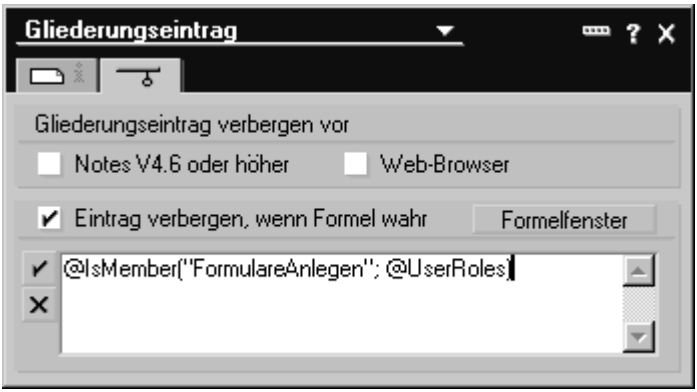


Gliederungseigenschaften

Mehr Eigenschaften finden sich bei den einzelnen Einträgen. Hier wird definiert, was geschehen soll, wenn der Benutzer einen Eintrag anklickt.

Eigenschaften des Gliederungseintrages

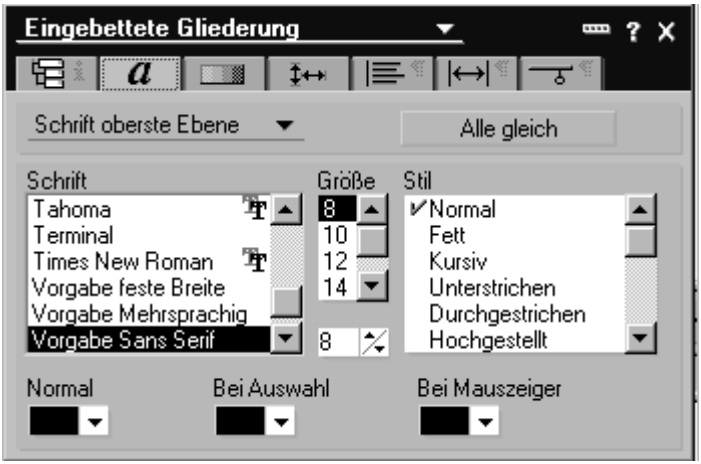
Die Festlegung des aufgerufenen Elementes entspricht der des Rahmens. Auch beim Eintrag kann wieder der Zielrahmen einer Rahmengruppe festgelegt werden. Vor einem Eintrag kann ein kleines Bild angezeigt werden, das über eine Bildressource oder eine Formel ermittelt wird. Der Auswahlfokus kann auf den aktuellen oder auf den vorherigen Gliederungspunkt eingestellt werden. Gliederungseinträge können, wie viele Gestaltungselemente, unter bestimmten Bedingungen verborgen werden.



Gliederungseintrag verbergen

In den Eigenschaften der Gliederung bzw. der Gliederungseinträge sucht man vergeblich nach einer Möglichkeit, die Schriftattribute der Menüpunkte zu verändern. Diese Einstellungen werden über das eingebettete Element „Gliederung“ auf der Seite eingestellt, in die die Gliederung eingefügt wird. Hier wird sie mit der rechten Maustaste ausgewählt und in dem Kontextmenü, das sich dann öffnet, die Position ”Eigenschaften Gliederung” angeklickt. In dem Eigenschaftsfenster hinter dem Register ”Schrift” kann die

Schrift für das Menü eingestellt werden. Dabei kann die Schrift für die Hauptpunkte und die Unterpunkte über eine Auswahl separat eingestellt werden.



Schriftart für Menüs

Nicht nur die Schriftart, sondern auch das ganze Aussehen der Gliederung wird auf der Seite bestimmt.

Eigenschaften der eingebetteten Gliederung

Neben dem Typ („Baumstruktur“ oder „Flache Struktur“) können eingestellt werden:

- Pfeilsymbol anzeigen

  Zeigt ein kleines Dreieck vor aufklappbaren Einträgen

- Titelstil

  Anzeige eines Titels über der Gliederung, um ein Rücksprungelement für die Untergliederungen zu erhalten.

- Zielrahmen

- Oberster Eintrag

  Zur Beschränkung der Anzeige auf die Unterpunkte eines Haupteintrages

- Größe

  Absolute oder relative Größe der Gliederung auf der Seite

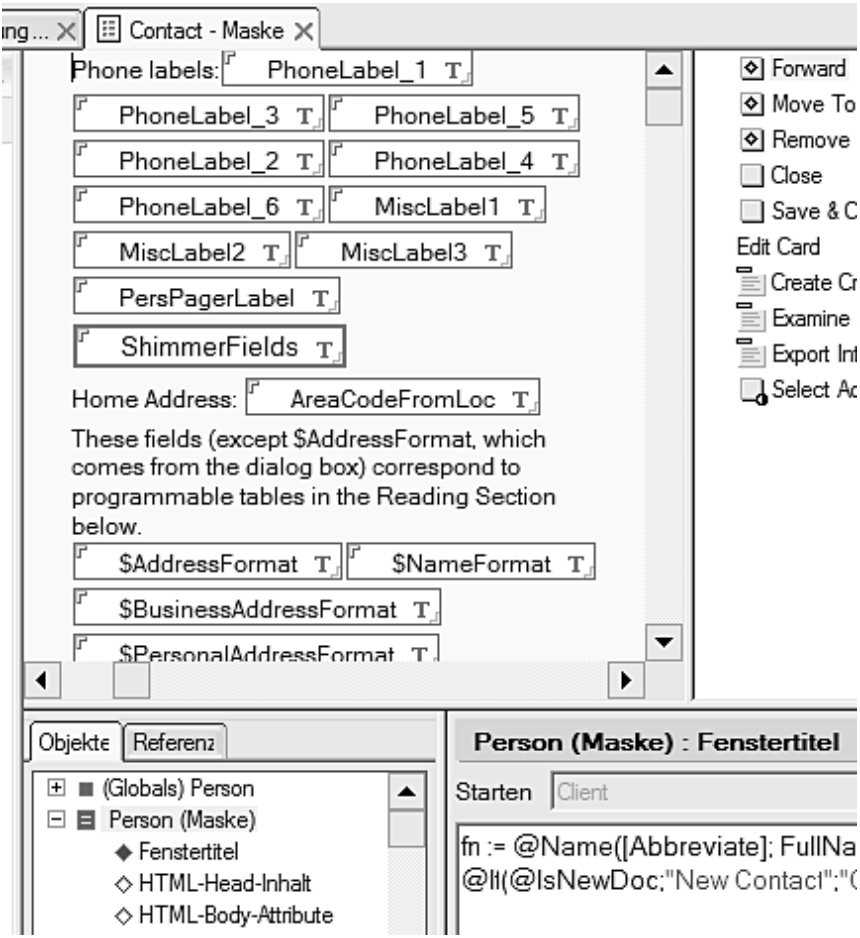
- Web-Zugriff

Bei Verwendung des Java-Applets sehen Gliederungen aus wie im Notes-Client. Die Verwendung setzt voraus, das die Browser Java-Applets anzeigen können.

Die fertige Seite mit der eingebetteten Gliederung kann jetzt in den Rahmen einer Rahmengruppe eingefügt werden.

4.3 Masken und Seiten

Erst die Maske macht Lotus Domino zu dem, was es unverwechselbar gemacht hat: einem Anwendungsserver, der es erlaubt, Client-/Serverlösungen zu erstellen und universell einzusetzen.



Maske im Designer

In Masken lassen sich Daten für Domino-Datenbanken erfassen, ändern und löschen. Sie ergänzen das E-Mail-System um Werkzeuge, die weit über den reinen Informationsaustausch hinausgehen.

Mit Release 5 wurde im Rahmen der Anpassung an Internet-Standards das Gestaltungselement Seite eingeführt.

Seiten im Lesezeichenordner des Designers

Die Seite entspricht in vielen Einzelheiten der Maske mit einer Ausnahme: eine Seite kann keine Felder, keine Layoutbereiche und keine Teilmasken aufnehmen. Damit ist die Seite ausschließlich in der Lage, Informationen anzuzeigen, während die Maske auch dazu dient, Daten zu bearbeiten.

Diese Seite ist zur Zeit nicht belegt.

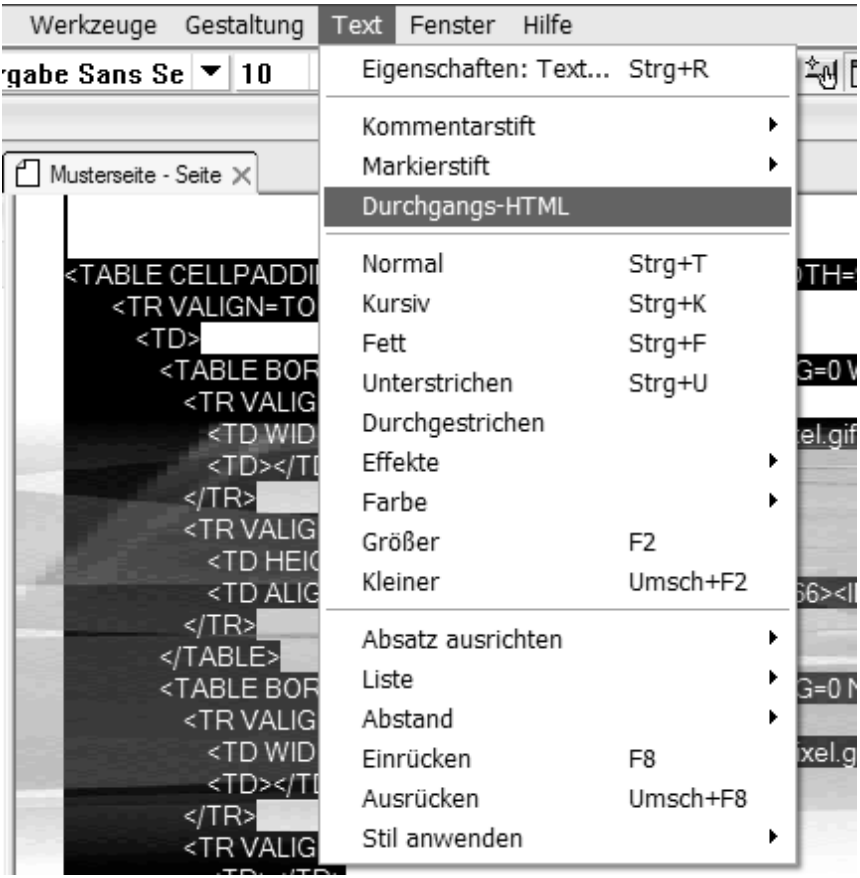
4.3.1 Seiten

Die Seite kann so aufgebaut sein, dass sie in Notes-Client und Browser nahezu gleich aussieht.

Seite im Notes-Client

Diese Möglichkeit wird noch dadurch unterstützt, dass HTML-Sprachelemente mit Release 6 auch im Notes-Client angezeigt werden. Der entsprechende Text wird dazu als Durchgangs-HTML gekennzeichnet. In Release 5 ist dieser Text nur im Browser an-

zeigbar. Der Notes-Client von Release 6 ist dagegen in der Lage, diesen HTML-Text zu rendern und anzuzeigen.

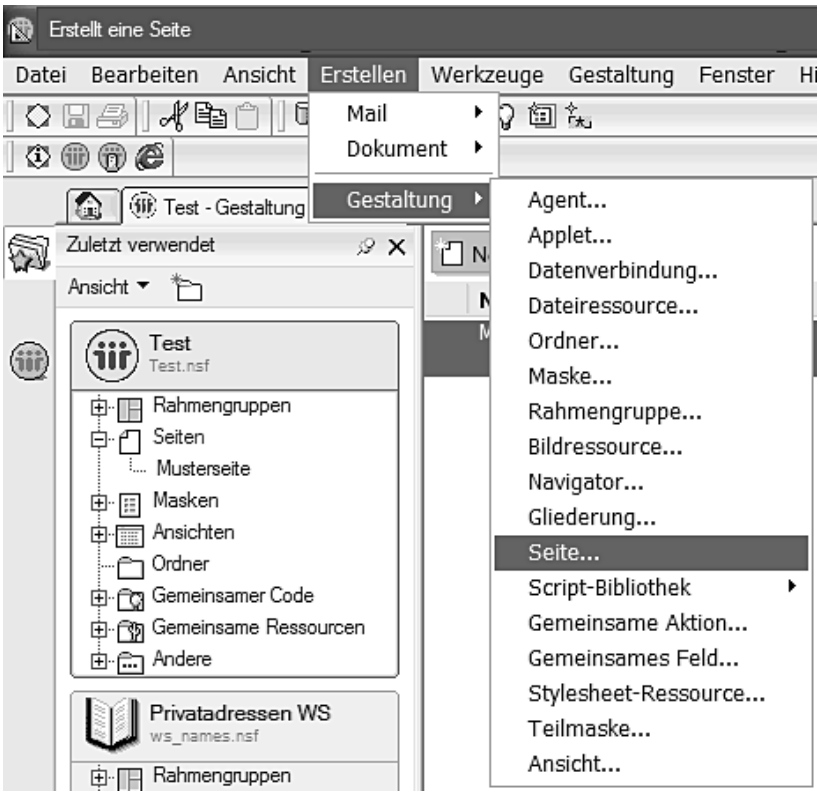


HTML-Text als Durchgangs-HTML

Eine Seite kann über die Menüposition „Erstellen / Gestaltung / Seiten“ erstellt werden. Es wird der Seiteneditor gestartet, der

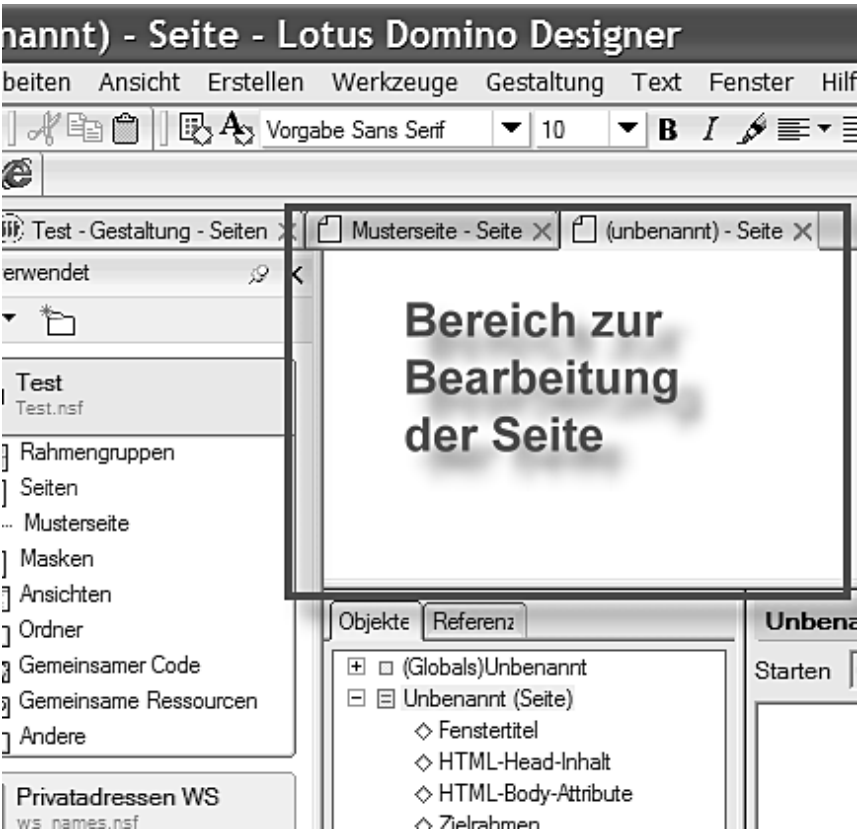
mit dem Maskeneditor identisch ist, aber nur die für Seiten verfügbaren Elemente zulässt. Der Editor ist viergeteilt und enthält die Bereiche:

- **Gestaltung**
- **Aktionen**
- **Objekte / Referenzen**
- **Programmtexte**



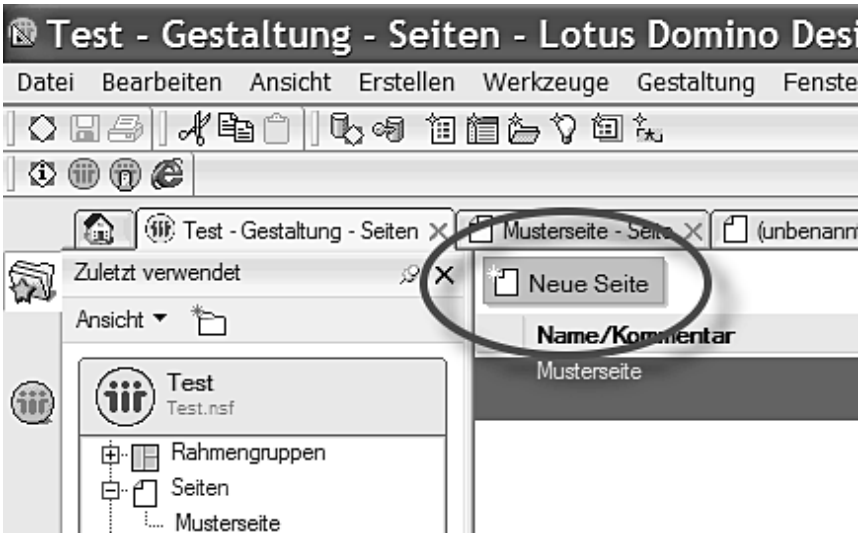
Neue Seite über das Menü erstellen

Im Gestaltungsbereich wird die Seite mit ihren verschiedenen Elementen angezeigt und bearbeitet. Bei einer neuen Seite ist dieser Bereich leer. Auch der Bereich der Aktionen enthält zunächst keine Elemente. Im Bereich „Objekte / Referenzen“ werden bereits die Standardobjekte für eine Seite aufgelistet und die Referenzen enthalten Unterstützung für Objekte und Programmiersprachen. Der Bereich für die Programmtexte zeigt nur leere Überschriften, wenn die LotusScript-Objekte (wie z.B. „PostOpen“) angeklickt werden.



Leere Seite

Eine weitere Möglichkeit, eine neue Seite zu erstellen bietet die Schaltfläche „Neue Seite“ in der Auflistung der Seiten einer Datenbank. Diese Auflistung wird im rechten Fenster des Designers angezeigt, wenn im Lesezeichenordner der Datenbank die Zeile „Seiten“ angeklickt wird.



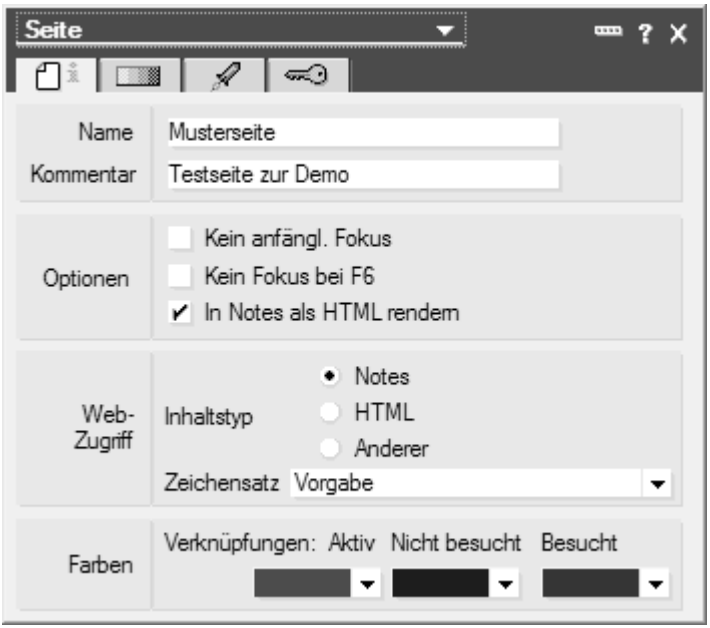
Die Schaltfläche „Neue Seite“

Neben den Angaben in dem Seiteneditor sind auch einige Eigenschaften der Seite in der Eigenschaftsdialogbox einzutragen. Diese Dialogbox wird über das Kontextmenü der rechten Maustaste oder das Symbol für Eigenschaften in der Symbolleiste des Designers aufgerufen. Die Register zeigen:

- **Allgemeine Eigenschaften der Seite**
- **Hintergrundeigenschaften**

- **Startoptionen**
- **Sicherheitseigenschaften**

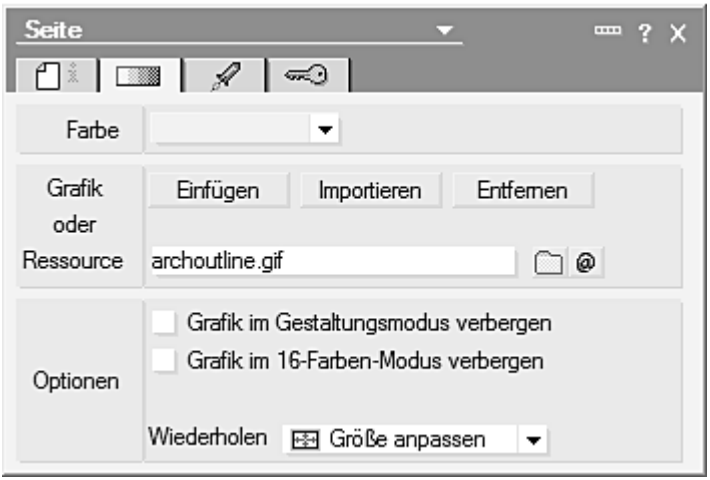
Im Register „Allgemein“ wird zunächst der Name der Seite festgelegt. Um von sprachlichen Eigenheiten unabhängig zu sein, empfiehlt es sich, für die Verwendung in Programmen einen so genannten Alias-Namen anzugeben. Der Alias-Name wird hinter den Namen geschrieben und von diesem durch ein „|“-Zeichen getrennt. Der Designer erkennt, dass hier ein Alias-Name vorliegt und verwendet ihn entsprechend. Zusätzlich kann im Kommentarfeld ein Text zur Beschreibung der Seite eingetragen werden.



Allgemeine Seiteneigenschaften

In weiteren Feldern wird festgelegt, ob in der Seite nach dem Start ein Cursor angezeigt wird, ob es sich bei dem Inhalt um Notes-Elemente, HTML-Text oder andere Inhalte handelt und welcher Zeichensatz für die Seite gelten soll. Sind in der Seite URLs, also Verknüpfungen enthalten, so lässt sich festlegen, welche Farben für aktive, nicht besuchte und besuchte Verknüpfungen verwendet werden.

Das Register für die Gestaltung des Hintergrundes enthält Angaben zu einer generellen Hintergrundfarbe, zu einer Hintergrundgrafik und zu weiteren Einstellungen. Die einfachste Form, den Hintergrund einer Seite zu gestalten, ist die, eine einheitliche Farbe als Hintergrund zu wählen. Diese Möglichkeit bringt häufig keinen Vorteil gegenüber einem weißen Hintergrund. Bedenkt man aber, dass eine Seite ein Element eines Rahmens in einer Rahmengruppe sein kann, dann ergeben sich mit dieser Variante bereits interessante Anzeigeformen.



Hintergrundeigenschaften der Seite

Hintergrundgrafiken sind natürlich wesentlich wirkungsvoller als reine Farbflächen. Es werden dazu eine ganze Reihe von Grafikformaten unterstützt:

- **BMP**
- **JPEG**
- **GIF**
- **PCX**
- **TIF**

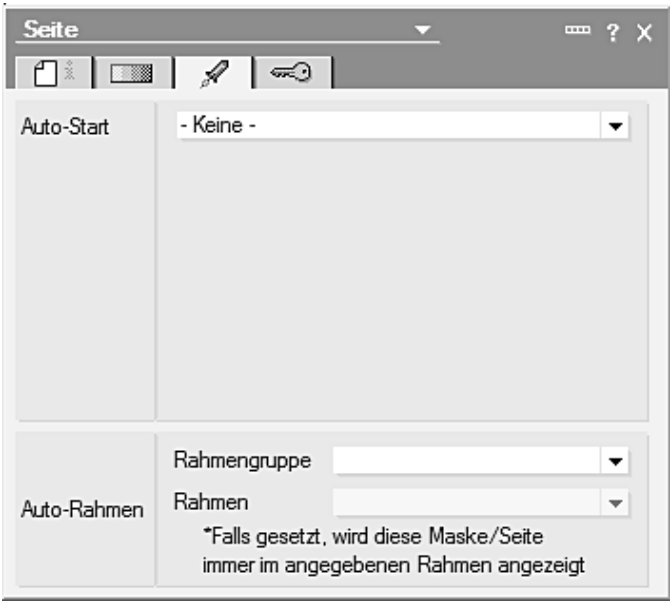
Die Grafiken können als Datei importiert oder über die Windows-Zwischenablage eingefügt werden. Je nach Größe der Grafik kann sie einmal oder mehrfach angezeigt werden. Folgende Varianten sind vorgesehen:

- **einmal**
- **vertikal**
  mehrfach untereinander
- **horizontal**
  mehrfach nebeneinander
- **Größe anpassen**
  Die Grafik wird automatisch an die Größe der Seite angepasst
- **zentriert**
  Die Grafik wird in der Mitte der Seite angeordnet

Je nach technischer Ausstattung des Entwicklerarbeitsplatzes kann es lästig sein, ständig auf die Anzeige der Hintergrundgrafik zu warten. Aus diesem Grund lässt sich die Grafik im Designer aus-

blenden und auch für die Anzeige bei Bildschirmen mit niedriger Auflösung unterdrücken.

Im Register „Starten“ kann ein Objekt, das in der Seite enthalten ist, automatisch gestartet werden.



Starteigenschaften der Seite

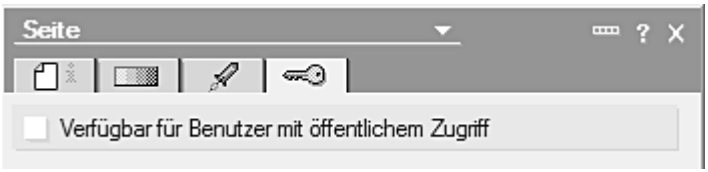
Ein solches Objekt kann sein:

- **ein Dateianhang**
- **eine Dokumentverknüpfung**
- **eine URL (Internetadresse)**
- **ein OLE-Objekt**

Achtung!

Diese Technik setzt allerdings voraus, dass der Benutzer über die entsprechenden Voraussetzungen zur Ausführung der Startaktion verfügt. So muss z.B. Microsoft Word installiert sein, wenn ein WORD-OLE-Objekt gestartet werden soll.

Die Seite kann in diesem Register fest mit einer Rahmengruppe und einem Rahmen verbunden werden. Diese Einstellung hat Vorrang vor einer Einstellung auf einer höheren Ebene (z.B. in einem Rahmen einer Rahmengruppe).



Sicherheitseigenschaften der Seite

Das Register „Sicherheit“ regelt, ob Benutzer mit öffentlichem Zugriff diese Seite verwenden dürfen. Ist dieses Recht angekreuzt, kann jeder Benutzer auf dieses Element zugreifen, auch wenn das in den Datenbank-Zugriffsrechten pauschal ausgeschlossen wurde sowie die übergeordnete Maske und Ansicht ebenfalls für öffentlichen Zugriff freigegeben wurden.

Die wichtigste Aufgabe der Seite besteht darin, andere Gestaltungselemente aufzunehmen und anzuzeigen. Neben einfachem bzw. gestaltetem Text können folgende Elemente in Seiten eingefügt werden:

- **Bilder**
- **berechneter Text**
  z.B. ein Datum oder ein Benutzername
- **Tabellen**
- **Eingebettete Objekte**
  Ansichten und Ordner, Gliederungen, Navigatoren und Datumswähler
- **Ebenen**
- **Java-Applets**
- **Verknüpfungen**
  zu Notes-Dokumenten oder Internet-Seiten

Die zweite Aufgabe der Seiten ist, Rahmen in Rahmengruppen anzuzeigen. Dadurch wird eine vielfältige Kombination von Anzeigeformen erlaubt.

Diese Seite ist zur Zeit nicht belegt.

4.3.2 Masken

Masken in Lotus Domino und Notes hängen eng mit den Dokumenten einer Datenbank zusammen. Sie dienen zur Erfassung, zur Anzeige und zum Drucken der Daten. Masken werden unabhängig von den Daten in eigenen Beschreibungsdokumenten gespeichert. Die Daten eines Dokumentes können mit unterschiedlichen Masken verbunden werden. Das bedeutet, dass eine Maske nicht unbedingt alle Felder eines Dokumentes anzeigt.

Der Name der Maske wird in dem Feld „Form“ gespeichert. Über diesen Namen wird das Dokument in der Datenbank mit der Maske verbunden.

- Befindet sich die Maske nicht in der Datenbank des Dokumentes, wird geprüft, ob die Maske mit dem Dokument gespeichert wurde.
- Ist das nicht der Fall, wird geprüft, ob in der Ansicht eine Formel zur Ermittlung der Maske zu finden ist.
- Als letzte Möglichkeit wird versucht, die Standardmaske der Datenbank zu verwenden.
- Ist das auch nicht möglich, wird eine Fehlermeldung ausgegeben und das Dokument nicht geöffnet.

Neben den Daten wird auch die an Maskenobjekte und -ereignisse gebundene Programmlogik in der Maske definiert und gespeichert. Dadurch werden Masken zu intelligenten Formularen, die zur Laufzeit (d.h. wenn der Benutzer arbeitet) Aktionen, Prüfungen und Meldungen auslösen können.

Eine Maske kann optisch wie ein Formular gestaltet werden. Dabei werden die Masken-Elemente so angeordnet, dass sowohl Funktion als auch Erscheinungsbild der Maske optimiert werden. Der Designer stellt neben Texten und Feldern eine ganze Reihe spezieller Elemente zur Verfügung. Die wichtigsten Elemente sind:

- **feststehender Text**
- **berechneter Text**
- **Erfassungs- und Anzeigefelder**
- **Tabellen**
- **Bilder**
- **Teilmasken**
- **Abschnitte**
- **Layoutbereiche**
- **Aktionen**
- **Hotspots**
  Schaltflächen, Anzeigefenster, Verknüpfungen
- **eingebettete Elemente**
  Gliederungen, Ansichten oder Ordner, Navigatoren, Datumswähler, Gruppenkalender, eingebettete Maske

- **Java-Applets**
- **Dateianhänge**
- **Horizontale Linien**
- **HTML-Text**
- **Image-Maps**
  Bilder mit hinterlegten Verknüpfungen
- **Ebenen**

Eine Maske kann auch dadurch erstellt werden, dass eine vorhandene Maske kopiert und abgeändert wird.

Maske

Die vorhandenen Masken werden in der Auflistung im rechten Fenster des Designers angezeigt. Seit Release 6 zeigt auch die linke Seite des Designerfensters einen Ausschnitt der vorhandenen Masken an, wenn das „+"-Symbol angeklickt wird. Über Pfeiltasten kann durch die Liste gescrollt werden.

In dem Eigenschaftsfenster befinden sich bei dem Objekt „Maske" die gleichen Registerabschnitte wie bei der Seite. Die einzelnen Register enthalten aber zusätzliche Felder und es wurden auch Abschnitte ergänzt. Es werden folgende Register mit Maskeneigenschaften angezeigt:

- **Allgemeine Eigenschaften**
- **Erweiterte Eigenschaften**
- **Startoptionen**
- **Hintergrundgestaltung**
- **Maskenkopf**
- **Druckeigenschaften**
- **Sicherheitseinstellungen**

Register in den Maskeneigenschaften

Im Verhältnis zur Seite werden bei der Maske wesentlich mehr Eigenschaften verwaltet. Neben dem Register „Allgemein“ wird noch ein Register „Erweitert“ angezeigt, um alle Eigenschaften bearbeiten zu können.

Da die Maske über ihren Namen den Dokumenten zugeordnet wird, muss sie über einen eindeutigen Namen verfügen. Notes berücksichtigt die Groß-/Kleinschreibung bei Namen. Der Name kann maximal 256 Zeichen lang sein.

Achtung!

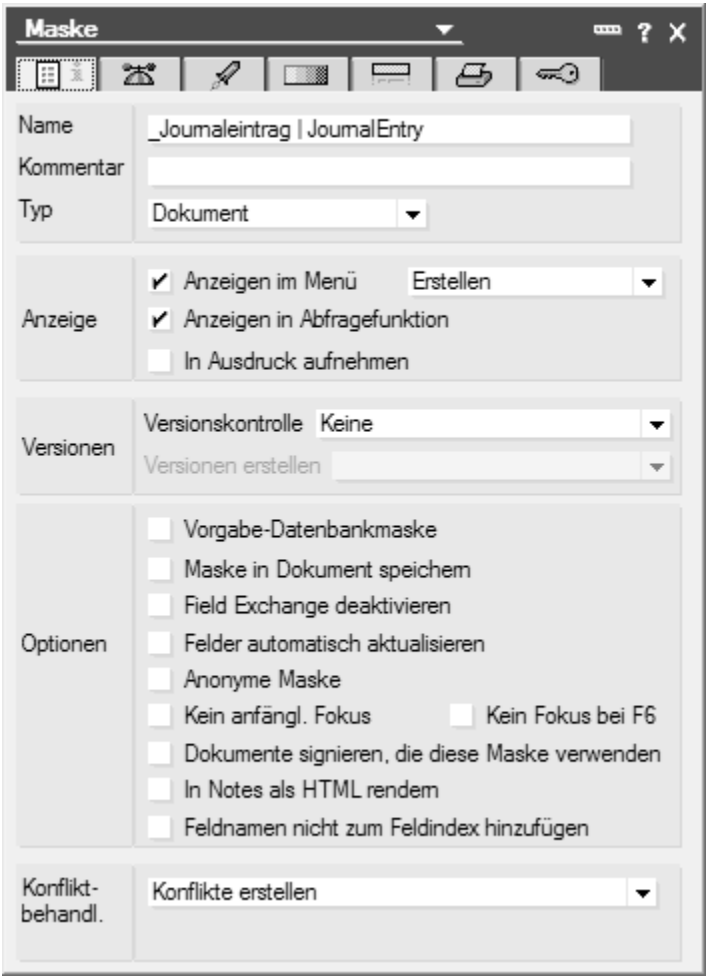
Da in dem Menü nur 64 Zeichen angezeigt werden, sollte er diese Länge normalerweise nicht überschreiten. Wie bei der Seite ist es auch bei der Maske möglich, einen Alias-Namen zu vergeben. Das ist besonders wichtig, wenn im Menü ein spezieller Name angezeigt werden soll. Dabei kann es erwünscht sein, einen Buchstaben zu unterstreichen, um ihn für die Tastenkombination „Alt + Buchstabe“ vorzubereiten. Damit ein Buchstabe im Menü unterstrichen wird, erhält der Maskenname einen Tiefstrich („_“) vor dem gewünschten Buchstaben (z.B. „_Kunde“ wird im Menü als „Kunde“ gezeigt).

Eine weitere Anforderung an die Menügestaltung stellt der Wunsch dar, ein Untermenü anzuzeigen. Für diesen Fall wird die Position des übergeordneten Menüs von der Position des Untermenüs durch ein Backslash („\“) getrennt (z.B. „Rechnung \ Kunde“).

In einem zusätzlichen Feld kann der Dokumententyp ausgewählt werden. Als Typen stehen „Dokument“, „Antwort“ und „Rückantwort“ zur Auswahl. Normalerweise wird der Standardwert „Dokument“ gewählt. Soll eine Maske für eine Diskussionsdatenbank oder eine ähnlich gelagerte Anwendung erstellt werden, so können auch die beiden anderen Typen zur Anwendung kommen.

Zum Erstellen eines Dokumentes stehen verschiedene Vorgehensweisen offen. Die häufigsten Formen sind:

- **Eine Menüposition im Menü „Erstellen“**
- **Eine Aktion in einer Ansicht**
- **Eine Verknüpfung in einer Seite oder Maske**



Allgemeine Maskeneigenschaften

In den Maskeneigenschaften kann definiert werden, ob die Maske im Menü „Erstellen“ aufgeführt werden soll.

Beim Suchen von Dokumenten kann die Maske für die Festlegung von Suchbegriffen verwendet werden. Soll der Benutzer die neue Maske für diesen Zweck einsetzen, dann wird das Kästchen „Anzeige in Abfragefunktionen“ angekreuzt.

Etwas ganz besonderes stellt das Feld „in Ausdruck aufnehmen“ dar. Dieses Feld ist neu in Release 6 und ermöglicht den Druck von Listen und Etiketten. Wenn die neue Maske in diesem Feld angekreuzt wird, kann sie als Basis für den Druck mehrer Dokumente auf eine Seite eingesetzt werden. Es sind dazu weitere Bedingungen zu erfüllen, die an anderer Stelle ausführlich erläutert werden.

Normalerweise werden alle Änderungen an einem Dokument direkt in diesem Dokument gespeichert. Da es möglich ist, dass zwei Benutzer dasselbe Dokument gleichzeitig verändern, kann es dadurch zu Speicherkonflikten kommen (d.h.: welche Änderung soll gelten?). Notes erstellt in diesem Fall ein zusätzliches Dokument mit dem Titel „Replizier- und Speicherkonflikt“. Eine Möglichkeit, solche Probleme zu umgehen stellt die Versionskontrolle dar. Diese Methode erstellt von vornherein für jede Änderung ein neues Dokument. Es kann gewählt werden, ob der Benutzer diese Entscheidung alleine treffen soll oder ob die getrennte Speicherung automatisch erfolgt.

Zur besseren Unterscheidung der Dokumentversionen kann eine der folgenden Speicherformen eingestellt werden:

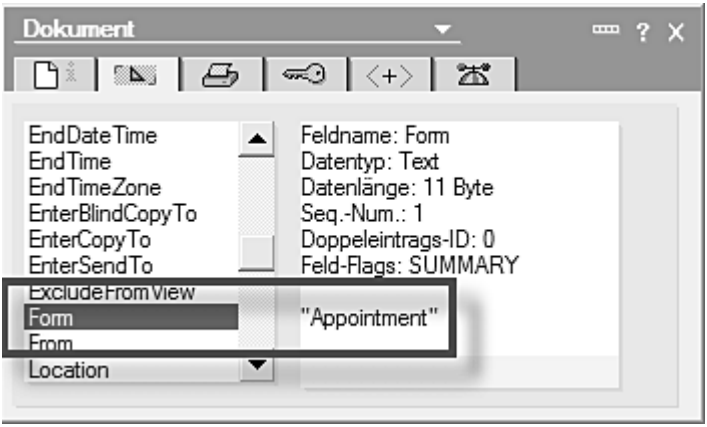
- **Neue Versionen werden Antworten**
  Das Originaldokument wird als Erstes in der Ansicht aufgeführt. Die Dokumente mit Änderungen werden darunter aufgelistet.
- **Frühere Versionen werden Antworten**
  Die zuletzt erstellte Version des Dokumentes wird als Hauptdokument geführt. Die älteren Versionen werden darunter gezeigt.

- **Neue Versionen werden Geschwister**
  Es werden keine Antwortdokumente, sondern nur Hauptdokumente erstellt. Das Originaldokument wird in der Ansicht als erstes Dokument angezeigt.

Wenn die Erstellung von Versionsdokumenten vom Benutzer ausgelöst werden soll, muss ein entsprechender Hinweis in der Anwendungsdatenbank erfolgen.

Weitere Optionen für die erstellten Dokumente und deren Maske sind:

- **Vorgabedatenbankmaske**
  Legt fest, dass diese Maske als Standardmaske für die Datenbank verwendet wird. Jede Datenbank kann nur eine Standardmaske enthalten. Diese Maske wird verwendet, wenn die Maske, deren Name im Feld „Form“ steht, nicht gefunden wird. Es ist empfehlenswert, in jeder Datenbank eine Maske als Vorgabemaske zu kennzeichnen.



Maskenname im Feld Form

- **Maske in Dokument speichern**
  Um ganz sicher zu gehen, dass die Maske immer für das Dokument zur Verfügung steht, kann hier festgelegt werden, dass die Maske im Dokument gespeichert wird. Da jetzt in jedem Dokument alle Informationen über den Aufbau der Maske gespei-

chert werden, muss mit einem entsprechenden Speicherbedarf für die Datenbank gerechnet werden. Eine nachträgliche Änderung der Maske wird nicht unbedingt in alle bestehenden Dokumente übernommen.

- **„Field Exchange" deaktivieren**
  Der Austausch von Feldinhalten erfolgt über einen OLE-Server. Das bedeutet, dass Felder mit gleichem Namen ihre Inhalte austauschen, wenn ein OLE-Objekt in ein Notes-Dokument eingebunden wird. Durch das Anhaken dieses Feldes wird der Automatismus der Funktion unterbunden.

- **Felder automatisch aktualisieren**
  Felder können neben normalen Daten auch Formeln und Scriptprogramme enthalten. Wenn der Wert eines Feldes aus einer Berechnung resultiert, erfolgt die Anzeige des neuen Inhaltes standardmäßig nur, wenn der Benutzer die Anzeige aktualisiert oder wenn die Anzeige über eine andere Funktion angepasst wird. Wenn die Funktion „Felder automatisch aktualisieren" angekreuzt wird, werden die Berechnungen der Felder automatisch vorgenommen, wenn ein Feld verlassen wird. Diese Berechnung kann je nach Maskenaufbau relativ schnell gehen oder lange dauern. Der Vorteil, dass der Benutzer sofort sieht, wie sich seine Eingabe auswirkt, wird durch den Nachteil erkauft, dass die Anwendung sehr langsam werden kann.

- **Anonyme Maske**
  Der Autor des Dokumentes wird nicht in der Maske gespeichert.

- **Kein anfänglicher Fokus / Kein Focus bei F6 (Release 6)**
  Bei der Anzeige von Seiten oder Masken in Rahmengruppen kann es erwünscht sein, keinen Cursor anzuzeigen, wenn die Maske geöffnet wird.

- **Dokumente signieren, die diese Maske verwenden (Release 6)**
  Dokumente, die mit dieser Maske erstellt werden, erhalten automatisch die Signatur des Erstellers.

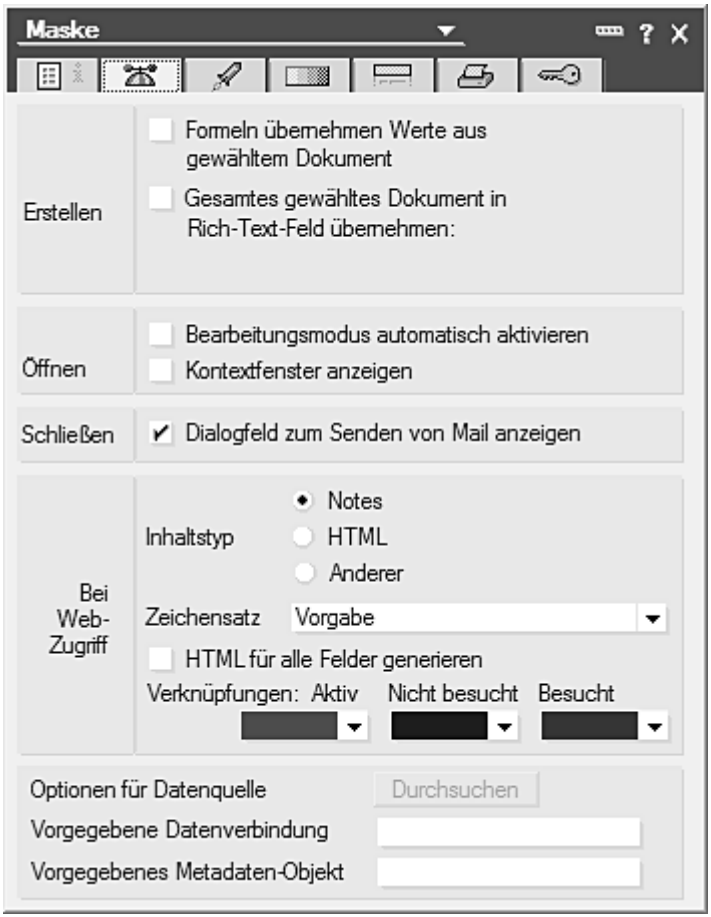
- **In Notes als HTML rendern (Release 6)**
  Erlaubt es, in der Maske mit PassThru-HTML zu arbeiten.

- **Feldname nicht zum Feldindex hinzufügen (Release 6)**
  Nimmt die Feldnamen in die nicht interne Feldliste auf. Damit stehen sie dem Benutzer z.B. bei der Erstellung von Agenten nicht zur Verfügung.

- **Replizierkonflikte mischen (nur bis Release 5)**
  Ist ein Dokument von zwei Benutzern gleichzeitig geändert worden und betreffen die Änderungen unterschiedliche Felder, so wird von Notes das Dokument in beiden Feldern geändert und kein Konflikt-Dokument erstellt, wenn diese Option eingeschaltet wird.

- **Konfliktbehandlung (Release 6)**
  In diesem Auswahlfeld kann festgelegt werden, ob
  - ein Konfliktdokument erstellt werden soll,
  - die Änderungen in einem Dokument gemischt werden sollen oder
  - nur eine der beiden Änderungen durchgeführt werden und die zweite entfallen soll.

Das zweite Register, die „Erweiterten Maskeneigenschaften“, beschäftigt sich mit Einzelheiten der Maskenbenutzung.



Weitere Maskeneigenschaften

Es enthält Bereiche, die regeln, was bei bestimmten Dokumentereignissen abweichend vom Standard geschehen soll.

- **Erstellen**
  Im Bereich „Erstellen“ kann geregelt werden, ob in einem neuen Dokument die Feldinhalte des vorher markierten Dokumentes übernommen werden sollen.

Die Feldinhalte werden nur dann übernommen, wenn die Namen der Felder des Ausgangsdokumentes und des neuen Dokumentes gleich sind.

Zusätzlich kann festgelegt werden, ob das markierte Dokument in ein Richtext-Feld des neuen Dokumentes übernommen werden soll.

- **Öffnen**
  Wenn ein Dokument geöffnet wird, ist es standardmäßig nicht editierfähig. Soll der Benutzer das Dokument direkt ändern können, muss das Feld „Bearbeitungsmodus automatisch aktivieren“ angekreuzt werden.
  Das Feld „Kontextfenster anzeigen“ bewirkt, dass in einem getrennten Fenster der Inhalt des „Eltern-Dokumentes“ angezeigt wird.

- **Schließen**
  Nur wenn es sich um ein e-Mail-Dokument handelt, ist es sinn-

voll, das Feld „Dialogfenster zum Senden von Mail anzeigen“ zu aktivieren. Dann wird nach Abschluss des Editiervorgangs gefragt, ob das Dokument verschickt oder nur gespeichert werden soll. Ein e-Mail-Dokument muss die für den Versand erforderlichen Felder (z.B. e-Mail-Adresse des Empfängers, Kopie, Absender usw.) enthalten.

- **Bei Web-Zugriff**
  Wird das Dokument über einen Browser bearbeitet, dann kann bestimmt werden, wie der Inhalt des Dokumentes behandelt werden soll:

  – als Notesdokument,
  – als HTML-Dokument oder
  – als ein benutzerdefiniertes Dokumentformat.

  Anstelle des Standardzeichensatzes kann dem Web-Benutzer erlaubt werden, einen anderen Zeichensatz aus einer Liste auszuwählen.

  Um verborgene Felder zum Speichern von Programmroutinen zu nutzen, kann das Feld „HTML für alle Felder generieren“ angekreuzt werden.

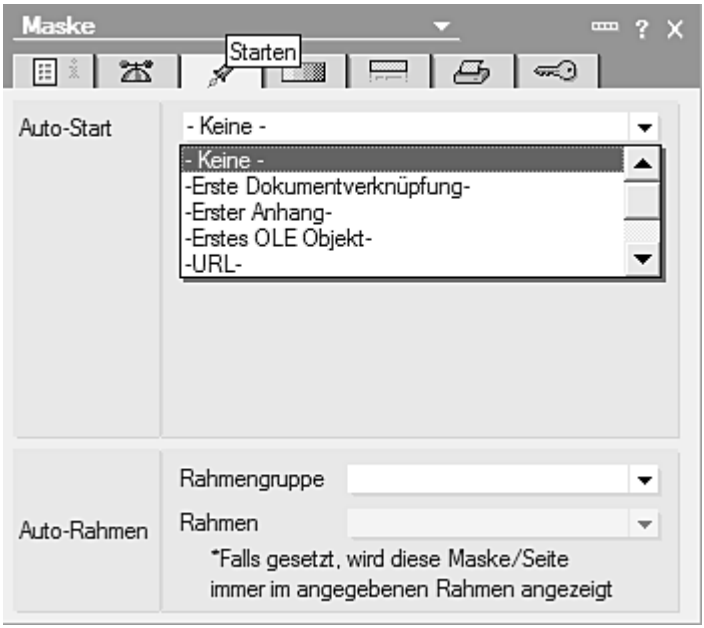
  Bei URLs können Farben für die Zustände:

  – aktiv,
  – nicht besucht,
  – besucht

  festgelegt werden.

- **Optionen für Datenquellen (Release 6)**
  Im Rahmen der Verwendung von externen Datenquellen als Ressourcen im Domino-Designer kann eine Maske direkt mit einer Datenquelle verbunden werden, die als Datenressource definiert wurde. Innerhalb der Maske stehen die Datenbankfelder dann als Felder zur Verfügung.

Das Register „Starten“ in den Maskeneigenschaften enthält Angaben zu Aktionen, die beim Starten eines Dokumentes, das diese Maske enthält, ausgeführt werden sollen.



Die Maskeneigenschaft „Starten“

Dabei unterscheidet der Designer zwischen Aktionen, die sich auf das Dokument selbst und solche, die sich auf den Rahmen und die Rahmengruppe beziehen, in dem das Dokument geöffnet werden soll.

- **Auto-Start**
  Beim Starten des Dokumentes kann statt des gesamten Dokumentes ein Objekt innerhalb des Dokumentes angezeigt werden:

  – **Erste Dokumentverknüpfung**
  Ist im Dokument eine Verknüpfung zu einem anderen Dokument gespeichert, so wird dieses Dokument angezeigt. Diese Funktion erlaubt es, eine Verzeichnis- oder Archiv-Datenbank zu erstellen, deren Dokumente auf unterschiedliche Dokumente aus unterschiedlichen Datenbanken verweisen.

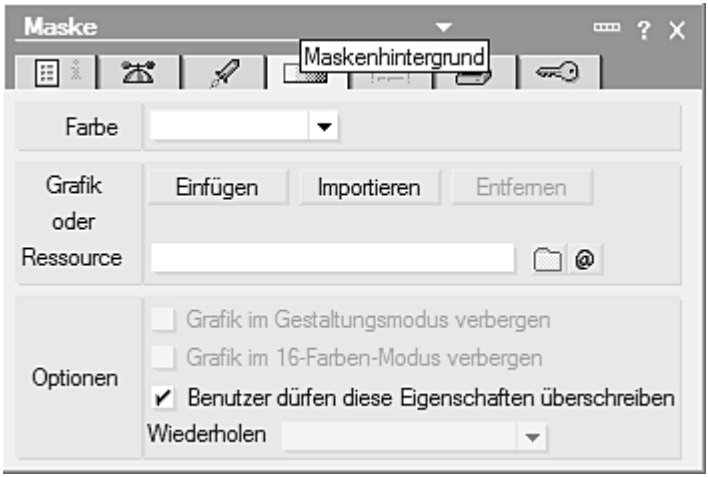
  – **Erster Anhang**
  Öffnet statt des Dokumentes den Anhang, der mit dem Dokument verknüpft ist. Dadurch können andere Dateiformate wie z.B. die Microsoft Word-Dokumente in Notes verwaltet, aber vom Benutzer mit den dazu gehörigen Programmen bearbeitet werden.

  – **Erstes OLE-Objekt**
  Mit dieser Auswahl wird festgelegt, dass beim Starten des Dokumentes das erste darin enthaltene OLE-Objekt mit dem dazugehörigen Programm gestartet wird.

  – **URL**
  Verzweigt beim Starten des Dokumentes zu der ersten im Dokument enthaltenen Internetadresse.

- **Auto-Rahmen**
  Legt fest, in welcher Rahmengruppe und in welchem Rahmen das Dokument geöffnet werden soll. Diese Definition überschreibt alle allgemeineren Zuordnungen des Dokumentes zu Rahmengruppen und Rahmen.

Für die Festlegung des Hintergrundes einer Maske ist ein eigenes Register verfügbar. Hier kann eine Hintergrundfarbe oder eine Grafik angegeben werden, die als Maskenhintergrund angezeigt wird. Außerdem kann festgelegt werden, ob der Benutzer der Maske diese Hintergrundeigenschaften verändern darf.



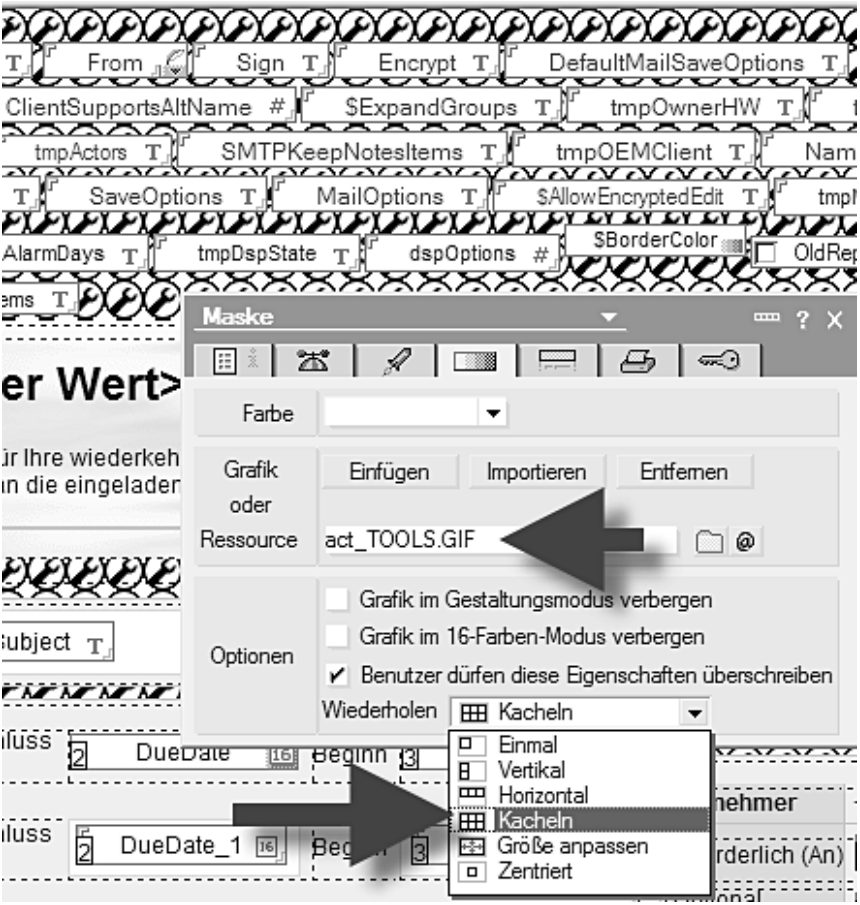
Der Maskenhintergrund

Soll eine Grafik verwendet werden, so bieten sich folgende Möglichkeiten:

- Es wird eine Grafik fest eingebunden, die als Ressource in der Datenbank verfügbar ist.
- Über eine Formel wird der Name einer Bild-Ressource ermittelt und diese auf diesem Weg als Grafik eingebunden.
- Über die Windows-Zwischenablage und die Schaltfläche "Einfügen" wird eine Grafik in die Maske eingebunden.
- Über die Schaltfläche „Importieren" wird eine Datei über den „Datei-öffnen"-Dialog ausgewählt und in die Maske eingebunden. Es werden die Grafikformate „BMP", „GIF", „JPEG", „PCX" und „TIFF 5.0" unterstützt.

Die Schaltfläche „Entfernen" erlaubt das Löschen einer Grafik aus dem Maskenhintergrund.

Über die Auswahl „Wiederholen" kann geregelt werden, ob die Grafik einmal oder mehrfach angezeigt werden soll.



Grafischer Maskenhintergrund

Folgende Versionen können ausgewählt werden:

- **Einmal**
  Die Grafik wird nur einmal, und zwar in der linken oberen Ecke der Maske angezeigt.

- **Vertikal**
  Die Grafik wird am linken Rand der Maske mehrfach übereinander angezeigt.

- **Horizontal**
  Die Anzeige der Grafik erfolgt am oberen Rand der Maske mehrmals nebeneinander.

- **Kacheln**
  Die Grafik wird sowohl horizontal als auch vertikal mehrfach angezeigt.

- **Größe anpassen**
  Die Größe der Grafik wird so verändert, dass sie den kompletten Hintergrund der Maske ausfüllt.

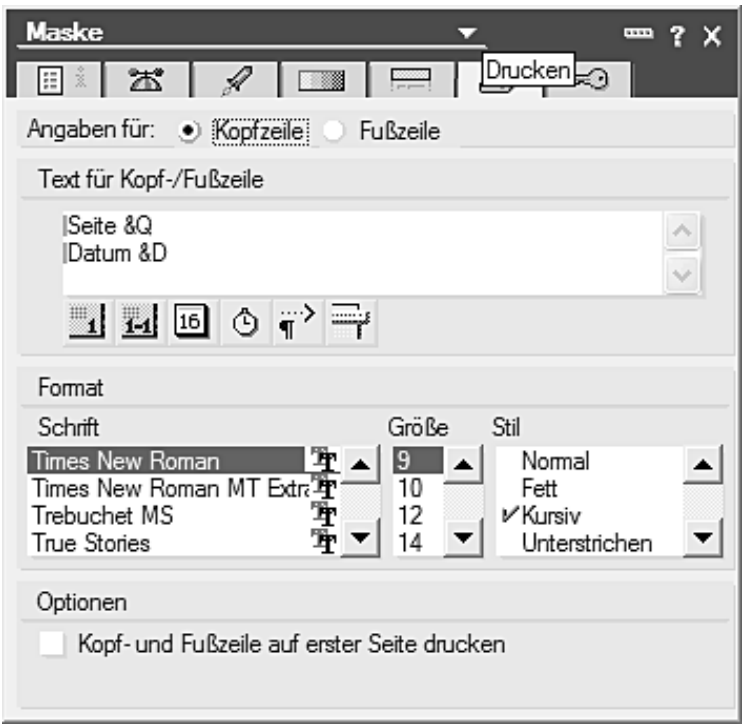
- **Zentriert**
  Die Grafik wird in der Mitte der Maske positioniert und nur einmal verwendet.

Bereits in der Maske können Angaben zu Kopf und Fuß eines Dokumentes im entsprechenden Register hinterlegt werden. In Release 5 ist zu dem gewohnten Druckregister eine neues hinzugekommen, das es erlaubt, einen feststehenden Bereich am Kopf des Dokumentes zu definieren, der fest stehen bleibt, wenn durch

das Dokument gescrollt wird. In diesem Bereich können Angaben gezeigt werden, die ständig verfügbar sein sollen – unabhängig davon, wie groß das Dokument ist.

Fester Dokumentenkopf

Das „Druck“-Register enthält Angaben zu Kopf und Fuß eines Dokumentes, die nur dann zu sehen sind, wenn das Dokument auf einem Drucker ausgegeben wird. Neben den Eigenschaften zur Schrift (je Bereich ist nur eine Schrift möglich) können noch spezielle Zeichen für Angaben eingesetzt werden, die automatisch durch Notes beim Druck eingefügt werden.



Druckeigenschaften

Die Sonderzeichen zur Ergänzung von Angaben durch Notes sind:

- **&P**
  Seitennummer

- **&Q**
  Seitennummer von Seiten

- **&D**
  Datum

- **&T**
  Uhrzeit

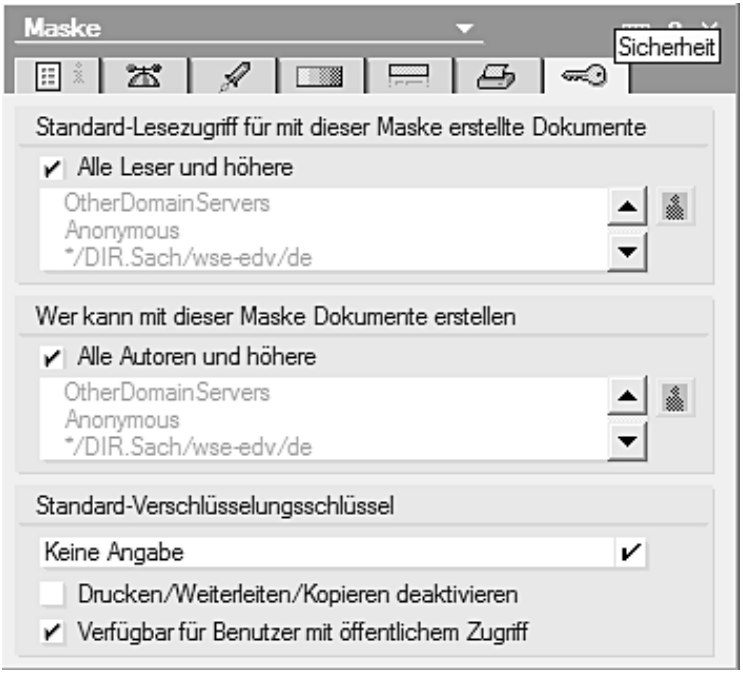
- **|**
  Tabulator zum Ausrichten am rechten oder linken Rand bzw. zum Zentrieren.

- **&W**
  Titel des Dokumentes

Die Sonderzeichen können über Schaltflächen eingefügt werden, die unter dem Textfenster angeordnet sind. Das Sonderzeichen wird dann jeweils am Ende des Textes angefügt.

Damit die Angaben nicht auf der Titelseite stehen, kann das Feld „Kopf- und Fußzeile auf der ersten Seite drucken“ deaktiviert werden.

Das letzte Register beschäftigt sich mit der Sicherheit und den Rechten, die für Dokumentenzugriffe gelten sollen.

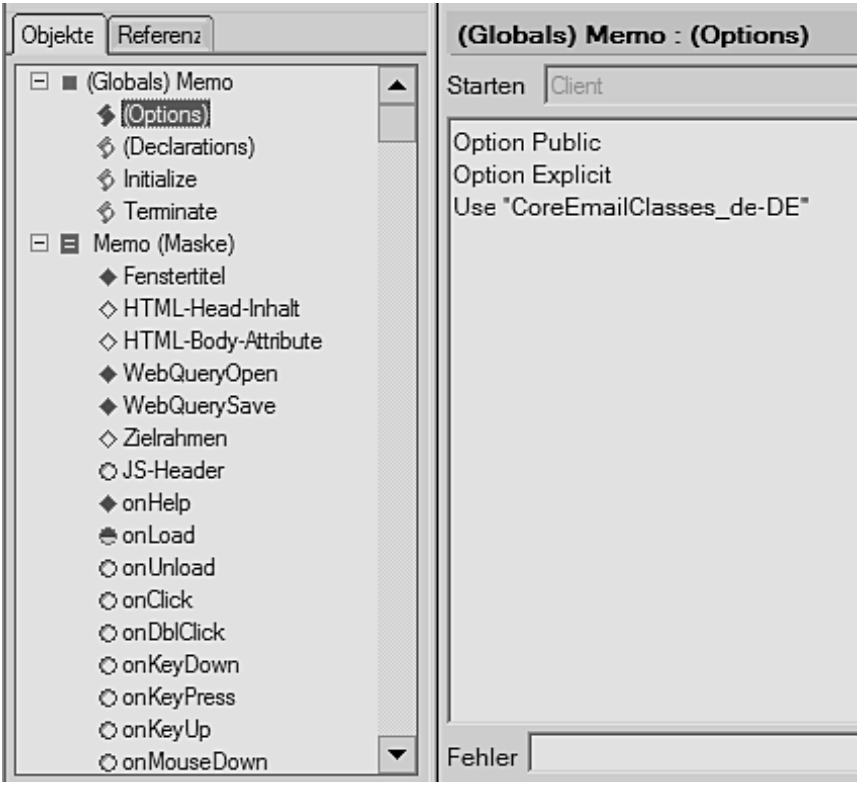


Dokumentensicherheit

In diesem Register kann geregelt werden, welche Rechte die Leser, die Autoren und die Benutzer mit öffentlichem Zugriff haben sollen.

Es kann festgelegt werden, ob ein Schlüssel zur Verschlüsselung der Dokumente eingesetzt werden soll (dabei muss der Schlüssel bereits definiert sein) und ob Dokumente, die mit dieser Maske erstellt wurden, gedruckt, weitergeleitet und kopiert werden können.

Eine weitere Möglichkeit, die Eigenschaften und das Verhalten der Maske zu steuern, besteht darin, die Maskenformeln und die Maskenereignisse einzusetzen. Diese Programmelemente werden in dem Domino-Designer im Programmierbereich definiert.



Maskenprogrammierung

Vor den Maskenereignissen können globale Variable, Start- und Endroutinen für die Maske definiert werden, die im Rahmen von LotusScript Verwendung finden.

Bei den Maskenereignissen können je nach Ereignis die Programmiersprachen – Formeln, LotusScript und Javascript – genutzt werden. Teilweise sind die Sprachen alternativ einsetzbar, teilweise kann auch nur eine der Sprachen zur Anwendung kommen. Ein Teil der Programmelemente sind für den Notes-Client, ein Teil für Browser-Zugriffe sowie ein Teil für beide Arten der Anwendung nutzbar. Hier zeigt sich der Vor- und Nachteil des Verfahrens, Anwendungen für unterschiedliche Zielsysteme mit einem Tool zu erstellen. Der Programmierer muss immer darauf achten, dass die Funktionalität, die er erzielen will, für den angestrebten Anwendungsbereich erreicht wird. Oft führt das dazu, dass Gestaltungselemente doppelt erscheinen – einmal für den Notes-Client und einmal für die Web-Anwendung. Das der Anwendung entsprechende Element wird sichtbar und das Gegenstück unsichtbar gemacht. In Release 6 werden neue Ereignisse in LotusScript eingeführt, die unter einem anderen Namen bereits verfügbar waren. So kann z.B. das „onLoad"-Ereignis mit Release 6 auch in LotusScript verwendet werden. Die bisherigen Namen (also in diesem Fall „PostOpen") können aus Kompatibilitätsgründen weiter benutzt werden. Die neuen Namen sind natürlich in älteren Releases nicht verwendbar.

Das erste Element in den Maskenereignissen stellt die Möglichkeit dar, einen Maskentitel zu definieren. Dieser Titel kann einfach als Zeichenkette (z.B. „Kunden") aber auch als Formel angegeben werden (z.B. „Brief erstellt am" + @Text(DateCreated)). An dieser Stelle zeigt sich die Problematik der Notes-Programmierung sehr deutlich. Die Formel kann ausschließlich in der Formelsprache angegeben werden. Als Notes-Programmierer kann man zwar die unterschiedlichsten Programmiersprachen verwenden (LotusScript, Java, Javascript), aber ohne die Formelsprache kommt man nie

aus. Die Angabe für den Fenstertitel wird im Notes-Client und im Web umgesetzt.

Die weiteren Maskenereignisse sind:

- **HTML-Head-Inhalt**
  Das Ereignis enthält HTML-Angaben, die zu den Anweisungen hinzugefügt werden, die der Domino-Server im <HEAD>-Bereich des Dokumentes automatisch generiert. Die HTML-Anweisungen können als Zeichenkette in Hochkommata oder als Ergebnis einer Formel eingetragen werden (z.B. „<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">").
  Sprachen: Formel
  Notes-Client: Nein
  Browser: Ja

- **HTML-Body-Attribute**
  HTML-Anweisungen, die in den <Body>-Bereich des Dokumentes eingefügt werden. Die HTML-Anweisungen können als Zeichenkette in Hochkommata oder als Ergebnis einer Formel eingetragen werden.
  Sprachen: Formel
  Notes.Client: Nein
  Browser: Ja

- **WebQueryOpen**
  Dieses Ereignis startet einen Agenten der im Rahmen einer Formel aufgerufen werden kann: (@Command [ToolsRunMacro]; „Name des Agenten");).
  Sprachen: Formel

Beispiel:
@If(@IsDocBeingEditedIreturnreceipt="1";@Command([ToolsRunMacro]; "wMemoOpen");"")
Notes-Client: Nein
Browser: Ja

- **WebQuerySave**
  Hier wird ein Agent gestartet, der ausgeführt wird, wenn Formeln berechnet und umgesetzt wurden sowie vor dem Speichern des Dokumentes auf die Festplatte. Dabei kann z.B. verhindert werden, dass das Dokument gespeichert wird, indem ein Feld mit dem Namen „Saveoptions“ den Wert „0“ erhält.
  Sprachen: Formel
  Beispiel: @Command([ToolsRunMacro]; "wMemo")
  Notes-Client: Ja
  Browser: Ja

- **Zielrahmen**
  Über die Formel des „Zielrahmen“-Ereignisses kann der Rahmen bzw. die Rahmengruppe festgelegt werden, in dem die Maske geöffnet wird.
  Sprachen: Formel
  Notes-Client: Nein
  Browser: Ja

- **JS-Header**
  In diesem Ereignis lassen sich Javascript-Funktionen hinterlegen, die der Domino-Server beim Erstellen der HTML-Seite automatisch einfügt. Die SCRIPT-Tags müssen nicht eingefügt werden. Domino fügt diese Tags selbständig in den HTML-Text ein.

Sprachen: Javascript
Beispiel:

```
var parentwnd = window.opener;

function LogoutCloseWin() {
  var strLocation = new String(parentwnd.parent.location);
  var str1 = new String(parentwnd.parent.location.protocol
  + "//" + parentwnd.parent.location.host + "/");
  var str2 = new String(strLocation.substring(str1.length,
  (strLocation.indexOf("nsf")+3)));
  var str3 = new String("?Logout");

  strLocation = str1 + str2 + str3;
  parentwnd.parent.location=strLocation;
  parentwnd.parent.close();
  self.close();
}
```

An dieser Position können auch Javascript-Bibliotheken eingefügt werden. Der Designer übernimmt die Bibliothek als Ressource und der Server fügt den Inhalt der Bibliothek beim Erstellen der HTML-Seite ein. Das Einfügen der Javascript-Bibliothek wird über das Menü „Erstellen“ vorgenommen. Die Position „Ressourcen“ startet einen Dialog, in dem die verfügbaren Javascript-Bibliotheken angezeigt und ausgewählt werden.

Javascript-Bibliothek einfügen

Im JS-Header-Ereignis wird jetzt ein Link-Symbol für die Javascript-Bibliothek angezeigt.
Notes-Client: Ja
Browser: Ja

- **onHelp(Release 6)**
  Dieses Ereignis tritt ein, wenn der Benutzer die „F1"-Taste drückt. Zur Auswertung stellt die Formelsprache den Befehl „OpenHelpDocument" zur Verfügung. In LotusScript kann über die „OpenPage"-Methode der Klasse NotesUiWorkspace eine Hilfeseite angezeigt werden.
  Das Ereignis „onHelp" ersetzt das Ereignis „Helprequest".
  Sprachen: Formel, LotusScript, Javascript

Beispiel Formel:
@If(@Subset(@DbName;1)="";@Command([OpenHelpDocument];[ClientHelp];"(Help)";"pernames_f_group");@Command([OpenHelpDocument];[AdminHelp];"(Help)";"pubnames_f_Group"))
Beispiel LotusScript:

```
Sub Onhelp(Source As Notesuidocument)
        Dim w As New NotesUIWorkspace
        Call w.OpenPage("HilfeSeite")
End Sub
```

Notes-Client: Ja
Browser: Ja (Javascript)

- **onLoad (Release 6)**
  Das „onLoad"-Ereignis tritt ein, wenn die Maske fertig geladen ist, die Eingabe durch den Benutzer aber noch nicht erlaubt ist. Das Ereignis kann deshalb dazu genutzt werden, Feldinhalte für den Benutzer vorzubereiten, die dynamisch zur Laufzeit berechnet sein sollen.
  Sprachen: Formel, LotusScript, Javascript
  Beispiel Javascript: document.forms[0].Status.value = "open"
  Notes-Client: Ja
  Browser: Ja (Javascript)

- **onUnload (Release 6)**
  Dieses Ereignis tritt auf, wenn das Dokument geschlossen werden soll. Über einen Indikator kann das Schließen abgebrochen und in den alten Zustand zurückgekehrt werden.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Ja (Javascript)

- **onClick. onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReset**
  Diese Ereignisse sind in Masken nur mit Javascript verbunden und können nur für Browser-Anwendungen genutzt werden.
  Sprachen: Javascript
  Notes-Client: Nein
  Browser: Ja

- **onSubmit (Release 6)**
  Das „onSubmit"-Ereignis tritt vor dem Speichern des Dokumentes ein. Im Gegensatz zum „onUnload"-Ereignis sind hier die Feldberechnungen und -prüfungen noch nicht ausgeführt. Über einen Rückgabewert kann das Speichern und Schließen des Dokumentes abgebrochen werden.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Ja (Javascript)

- **(Options)**
  Über Optionen lassen sich bestimmte Vorgaben für den LotusScript-Compiler definieren. So kann z.B. über die Option „Declare" festgelegt werden, dass alle Variablen definiert sein müssen, bevor sie in Anweisungen verwendet werden.
  Sprachen: LotusScript
  Notes-Client: Ja
  Browser: Nein

- **(Declarations)**
  Bei den Declarations können Variablen hinterlegt werden, die für die Maskenereignisse als „Global" gelten.

Sprachen: LotusScript
Notes-Client: Ja
Browser: Nein

- **QueryOpen**
  Das „QueryOpen"-Ereignis tritt ein, wenn ein Dokument geöffnet wird, aber für den Benutzer noch nicht verfügbar ist. Es liefert eine Reihe von Werten, die der Programmierer dazu verwenden kann, das Öffnen des Dokumentes zu verhindern, wenn einer dieser Werte nicht der vorgesehenen Verwendung des Dokumentes entspricht. Die Werte sind:

  – **Mode**
  Das Dokument wird zum Lesen oder Schreiben geöffnet.

  – **IsNewDoc**
  Das Dokument wurde gerade erstellt bzw. war bereits vorhanden.

  – **Continue**
  Dieser wert wird vom Programmierer auf „False" gesetzt, wenn der Benutzer das Dokument nicht öffnen soll.

  Sprachen: LotusScript
  Notes-Client: Ja
  Browser: Nein

- **PostOpen**
  Das Ereignis „PostOpen" hat die gleiche Wirkung wie das mit Release 6 eingeführte Ereignis „onLoad".
  Sprachen: LotusScript
  Notes-Client: Ja
  Browser: Nein

- **Querymodechange**
  Dieses Ereignis tritt ein, wenn der Status des Dokumentes von „Lesen“ auf „Schreiben“ gesetzt wird und umgekehrt. Es kann z. B. dazu genutzt werden, den Benutzer daran zu hindern, in den Editiermodus zu wechseln.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Nein

- **Postmodechange**
  Bevor der Benutzer wieder auf das Dokument zugreifen kann, tritt nach einem Statuswechsel dieses Ereignis ein. Soll z.B. zu einem Feldwert ein Protokoll geführt werden, so kann über dieses Ereignis ein Feld mit einem Textwert ergänzt werden, bevor der Benutzer mit dem Dokument arbeitet.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Nein

- **Queryrecalc (Release 6)**
  Wird eine Neuberechnung aller Feldwerte ausgelöst, so tritt das Ereignis „Queryrecalc“ ein, bevor die Neuberechnung durchgeführt wird. Sie kann über einen Schalter an dieser Stelle abgebrochen werden. Das ist z.B. dann denkbar, wenn Felder noch keine Werte enthalten.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Nein

- **Postrecalc**
  Nach Ausführung einer Neuberechnung tritt das Ereignis „Postrecalc“ ein, und zwar direkt bevor die Steuerung wieder an den Benutzer übergeht.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Nein

- **Querysave**
  Das „QuerySave“-Ereignis wird vor dem Speichern ausgeführt. Es wird in Release 6 durch das Ereignis „onSubmit“ ersetzt.
  Sprachen: Formel, LotusScript
  Notes-Client: Ja
  Browser: Nein

- **Postsave**
  Dieses Ereignis tritt nach dem Sichern des Dokumentes ein.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Nein

- **Querysend Release 6)**
  Wird ein Dokument als e-Mail versendet, durchläuft es beim Versand das „Querysend“-Ereignis, bevor Feldberechnungen und -prüfungen durchgeführt werden. Der e-Mail-Versand kann an dieser Stelle abgebrochen werden.
  Sprachen: Formel, LotusScript, Javascript

Beispiel:

```
Sub Querysend(Source As Notesuidocument, _
    Continue As Variant)

    Dim antw As Integer

    antw = Msgbox("Dokument senden?", _
      MB_YESNO, Source.FieldGetText("Subject"))
    If(antw = IDYES) Then
        Continue = True
    Else
        Continue = False
  End If

End Sub
```

Notes-Client: Ja
Browser: Nein

- **Postsend (Release 6)**
  Nach dem Versand eines Dokumentes als e-Mail wird das „Postsend"-Ereignis ausgelöst.
  Sprachen: Formel, LotusScript, Javascript
  Notes-Client: Ja
  Browser: Nein

- **Queryclose**
  Das „Queryclose"-Ereignis tritt ein, bevor ein Dokument geschlossen wird. Das Schließen kann über dieses Ereignis abgebrochen werden. Das Ereignis „Queryclose" entspricht dem „onUnload"-Ereignis in Release 6.

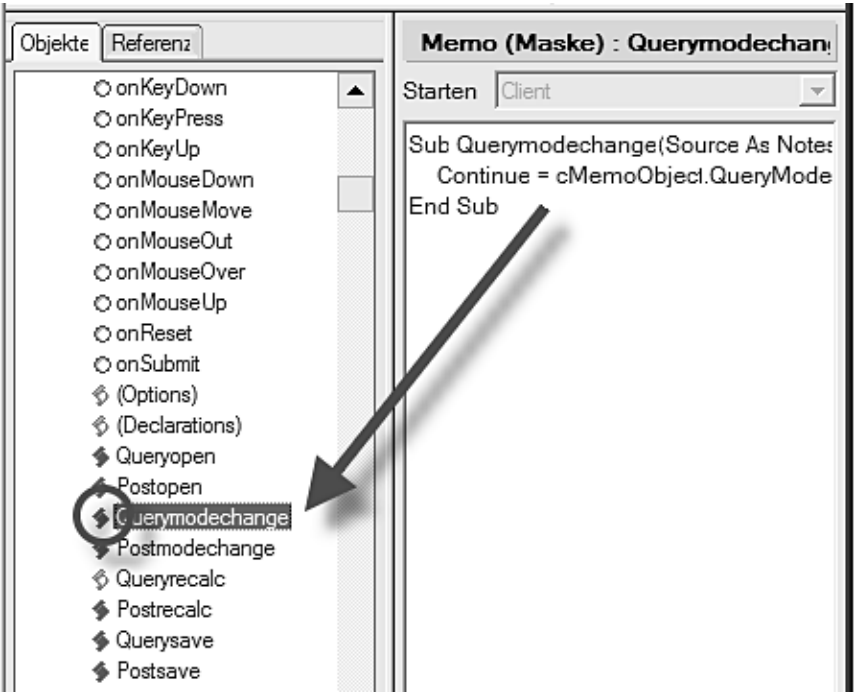
Sprachen: Formel, LotusScript
Notes-Client: Ja
Browser: Nein

- **Initialize**
  Das „Initialize"-Ereignis wird ganz zum Beginn des Ladevorganges eines Dokumentes durchlaufen. Dadurch stehen zu diesem Zeitpunkt erst wenige Informationen zur Verwendung bereit.
  Sprachen: LotusScript
  Notes-Client: Ja
  Browser: Nein

- **Terminate**
  Wenn das „Terminate"-Ereignis eintritt, kann bereits kein Einfluss mehr auf den Schließvorgang des Dokumentes genommen werden. Lediglich Informationen an den Benutzer in Form von Meldungen sind noch denkbar.
  Sprachen: LotusScript
  Notes-Client: Ja
  Browser: Nein

Die Abfolge der Ereignisse liegt beim Öffnen bzw. Schließen eines Dokumentes genau fest. Die Entscheidung darüber, welches Ereignis der Programmierer wählt, um ein bestimmtes Resultat zu erzielen, häng davon ab, welche Informationen zur Zeit des Ereignisses verfügbar sind und ob Inhalte des Dokumentes zu diesem Zeitpunkt geändert werden können.

Wird eines der Ereignisse mit Programmcode versehen, ändert sich das Symbol vor dem jeweiligen Ereignis.



Ereignissymbole

Der Programmierer kann also sofort erkennen, welche Ereignisse solche Programmquellen enthalten. Der ursprüngliche Bezug des Aussehens dieser Symbole zur Programmiersprache ist dadurch verloren gegangen, dass die Ereignisse Quellcode zu unterschiedlichen Programmiersprachen enthalten können.

Diese Seite ist zur Zeit nicht belegt.

4.3.3 Masken- und Seitenelemente

Sobald die Maske als Grundelement angelegt wurde, kann sie Felder, Texte und viele andere Elemente aufnehmen. Sie gleicht darin zunächst einmal der Seite. Der wesentliche Unterschied zur Seite besteht darin, dass die Maske vom Benutzer mit Informationen gefüllt werden kann und diese wieder in Datenbanken gespeichert werden können.

Die Entwickleroberfläche ist an Zeilen orientiert und die Objekte werden nicht wie bei anderen visuellen Umgebungen grafisch über Koordinaten positioniert. Die verfügbaren Gestaltungselemente wurden von Text- und Eingabefeldern zu einer großen Bandbreite an Möglichkeiten ausgebaut. Der Domino-Designer bietet für die Gestaltung eine ganze Serie von Elementen, die teilweise sogar alternativ eingesetzt werden können:

- **Text**
  Für die Erfassung von Texten kann im Designer einfach in das Bearbeitungsfenster geschrieben werden. Die Platzierung in der Maske erfolgt durch Zeilenschaltungen, Tabulatortasten und Leerzeichen.

- **Felder mit unterschiedlichen Eigenschaften**
  Als Feldtypen kennt der Designer einfache Erfassungsfelder und spezielle Elemente wie Options- und Auswahlfelder.

- **Ebene (Release 6)**
  Aus der Welt der Web-Programmierung wurde im Release 6 das Element „Layer“ bzw. „Ebene“ übernommen. Damit er-

hält der Programmierer eine Möglichkeit, Gestaltungselemente auf den Punkt genau innerhalb einer Maske zu platzieren.

- **Layoutbereich**
  Die Layoutbereiche wurden als Einstieg in eine vollständig grafisch orientierte Entwicklungsumgebung eingeführt. Mit dem Aufkommen der Webseitengestaltung und der Einbindung dieser Methoden und Verfahren in den Designer wurden die Layoutbereiche in den Hintergrund gedrängt. Man kann heute davon ausgehen, dass diese Elemente nicht mehr verwendet werden sollten.

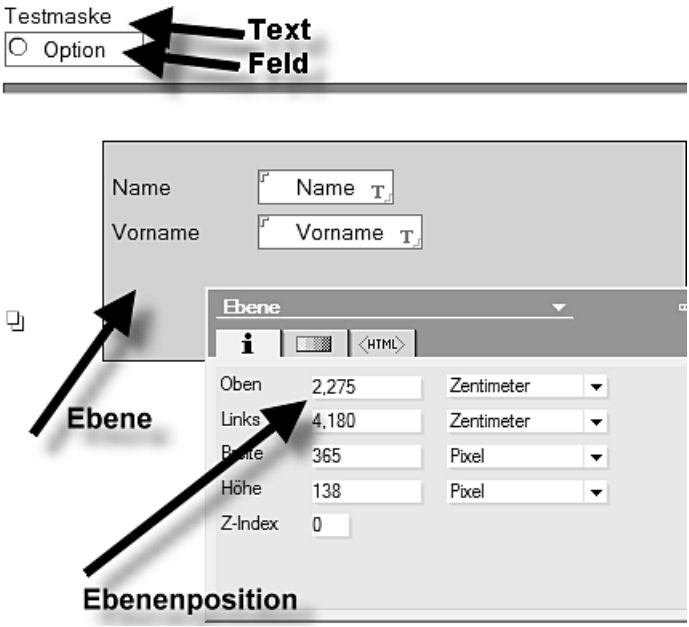
- **Aktion**
  Zur Interaktion mit dem Benutzer stehen Schaltflächen und Symbole zur Verfügung, die wie Menüs eingesetzt werden können. Aktionen können innerhalb einer Maske oder wieder verwendbar als Ressourcen gespeichert werden.

- **Ressource und Teilmaske**
  Um den Programmierer bei der Arbeit zu unterstützen, wurden eine ganze Reihe von Gestaltungselementen so angelegt, dass sie außerhalb der Maske in einem Bibliothekssystem zu finden sind. Dazu gehören unter anderem gemeinsame Felder und Aktionen, Teilmasken, Bilder und Programm-Bibliotheken.

- **Seitenumbruch**
  Die Maske kann in mehrere Seiten aufgeteilt sein, die z.B. über Verknüpfungen angesteuert werden.

- **Horizontaler Strich**
  Dieses Hilfsmittel dient einfach der vertikalen Untergliederung einer Maske oder Seite.

- **Abschnitt**
  Die Maske lässt sich in einzelne Abschnitte einteilen, die bestimmte Eigenschaften und Steuerelemente aufweisen.



Maske

- **Berechneter Text**
  Mithilfe von berechnetem Text können Informationen aus der Datenbank oder dem Systemumfeld zur Laufzeit mit aktuellen Werten angezeigt werden.

- **HotSpot**
  Der „HotSpot“ stellt eine Möglichkeit dar, durch Anklicken eines Elementes zu einer Verknüpfung zu verzweigen oder eine Aktion auszulösen.

- **Tabelle**
  Tabellen können dazu dienen, Texte und Felder innerhalb einer Maske zu positionieren. Die Varianten der Tabelle sind seit Release 5 sehr vielseitig und reichen bis zu animierten und programmgesteuerten Formen.

- **Bild**
  Bilder können in Masken an beliebiger Position eingefügt werden. Sie können auch als Image-Maps, also zur Steuerung von Verknüpfungen verwendet werden.

- **Bildressource**
  Statt Bilder direkt in einer Maske zu speichern, können sie auch als Ressource abgelegt werden. In diesem Fall stehen sie wie in einer Bibliothek zur mehrfachen Verwendung bereit.

- **Anhang**
  Soll jedes Dokument, das mit Hilfe einer bestimmten Maske erstellt wird, bereits mit einem Anhang versehen sein (wie z.B. einem Word-Dokument mit allgemeinen Geschäftsbedingungen), so kann dieser Anhang bereits in der Maske eingefügt werden.

- **Objekt**
  Auch OLE-Objekte können bereits in einer Maske eingebunden sein.

- **Java-Applet**
  Kleine Javaprogramme können in Masken bestimmte Funktionen übernehmen. Sie sind auch bei den Ressourcen speicherbar.

- **Eingebettetes Element**
  Über die „Eingebetteten Elemente“ lassen sich vorgefertigte Module des Designers auch als eigene Elemente in einer Maske verwenden. Solche Elemente können z.B. sein:

– Gliederung
– Ansicht
– Navigator
– Datumswähler
– Gruppenterminplaner
– Editor
– Ordnerfenster
– Schaltfläche „Datei laden“

Welche Elemente in einer Maske verwendet werden, hängt von der Anwendung ab sowie davon, in welchem Umfeld sie genutzt wird. Eine Anwendung, die ausschließlich über Browser bedient wird, muss zwangsläufig andere Maskenelemente einsetzen als solche Anwendung, die nur über Notes-Clients aufgerufen wird. Da immer mehr der Fall auftritt, dass Anwendungen über beide Client-Formen bedienbar sein sollen, werden teilweise sogar Elemente doppelt in der Maske eingesetzt. Die jeweils nicht verwendete Variante wird dann über die Eigenschaften des Elementes ausgeblendet.

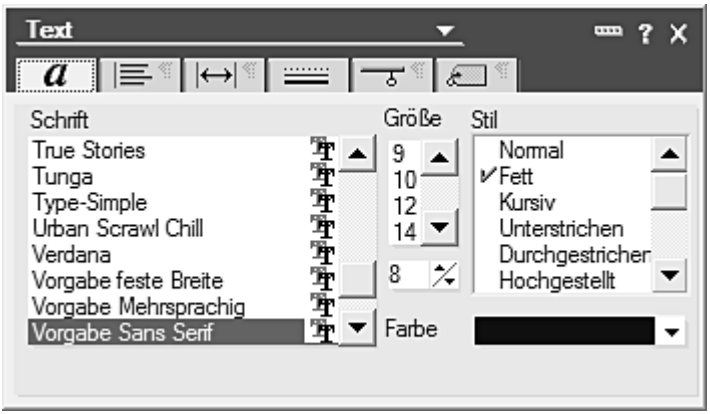
4.3.3.1 Felder und Texte

Sobald die Maske als Grundelement angelegt wurde, kann sie Felder, Texte und viele andere Elemente aufnehmen. Sie gleicht darin zunächst einmal der Seite. Der wesentliche Unterschied zur Seite besteht darin, dass die Maske vom Benutzer mit Informationen gefüllt werden kann und diese wieder in Datenbanken gespeichert werden können.

Normale Texte werden einfach in das Bearbeitungsfenster geschrieben. Die Platzierung in der Maske erfolgt durch Zeilenschaltungen, Tabulatortasten und Leerzeichen. Die Entwickleroberfläche ist textorientiert und die Objekte werden nicht wie bei anderen visuellen Umgebungen grafisch über Koordinaten positioniert. Texte dienen der Beschreibung von Feldern, können Hinweise enthalten und sind als berechnete Texte dynamisch aktualisierbar. Felder dienen der Maskensteuerung und der Informationsverwaltung.

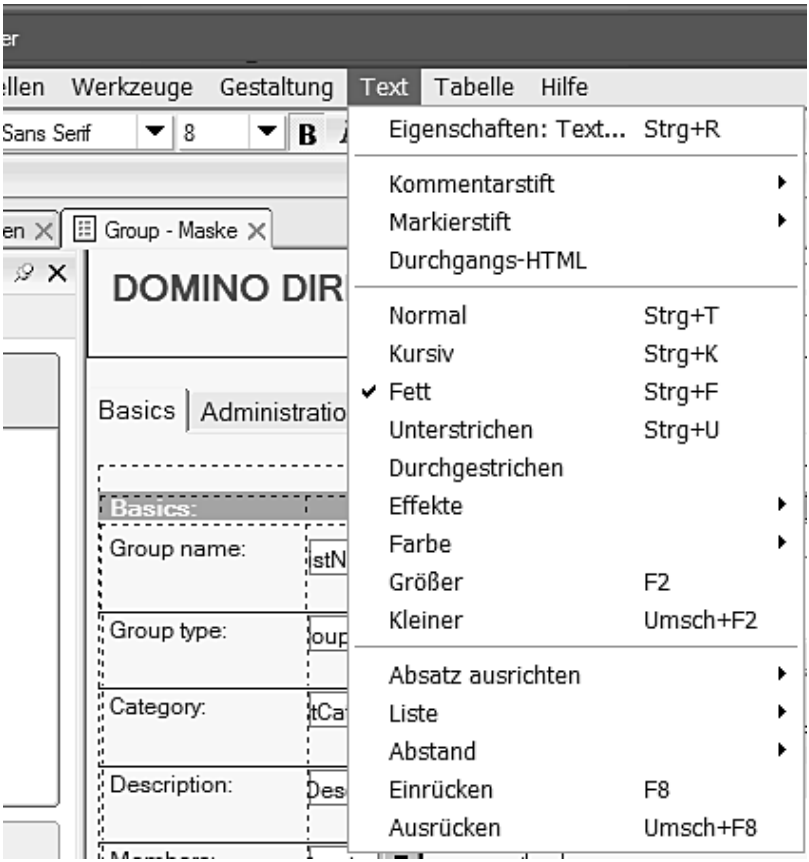
4.3.3.1.1 Text

Statischer Text kann einfach als Information in einer Maske angezeigt werden. Es können alle Möglichkeiten des Editors zur Gestaltung dieses Textes genutzt werden.



Texteigenschaften

Dazu wird entweder das Menü „Text“ im Designer oder das Eigenschaftsfenster eingesetzt.

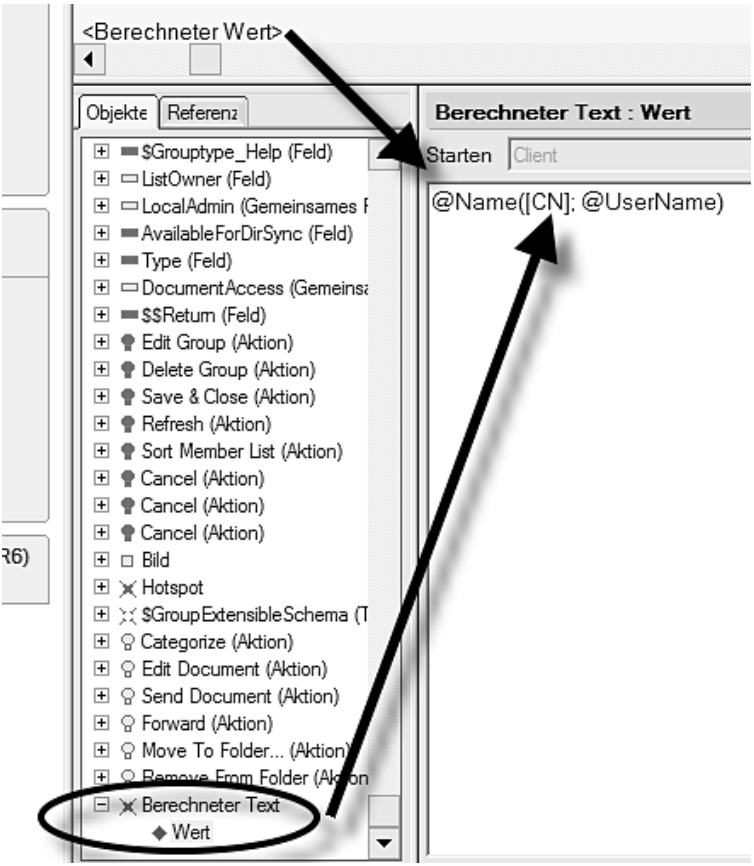


Das Textmenü

Der zu ändernde Text wird markiert und dann die gewünschte Eigenschaft ausgewählt. Sowohl die Schrift- und Zeichenattribute als auch die Absatzeigenschaften können zur Gestaltung der Mas-

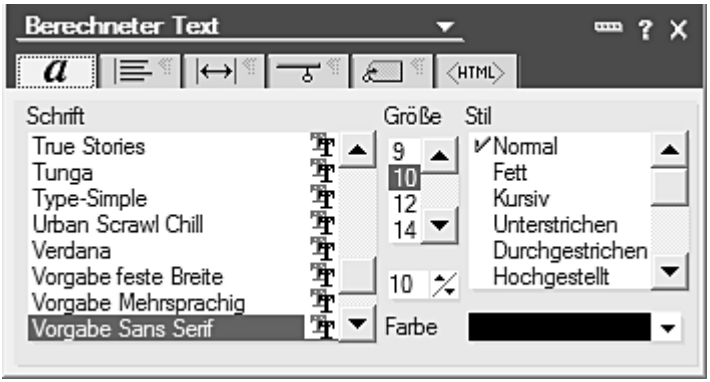
ke verwendet werden. Andere Gestaltungselemente lassen sich zwischen Texte einfügen. Sie beeinflussen allerdings den Zeilenumbruch.

Eine besondere Form des Textes stellt der berechnete Text dar. In der Maske wird ein Feld mit dem Text „<Berechneter Wert>“ angezeigt.



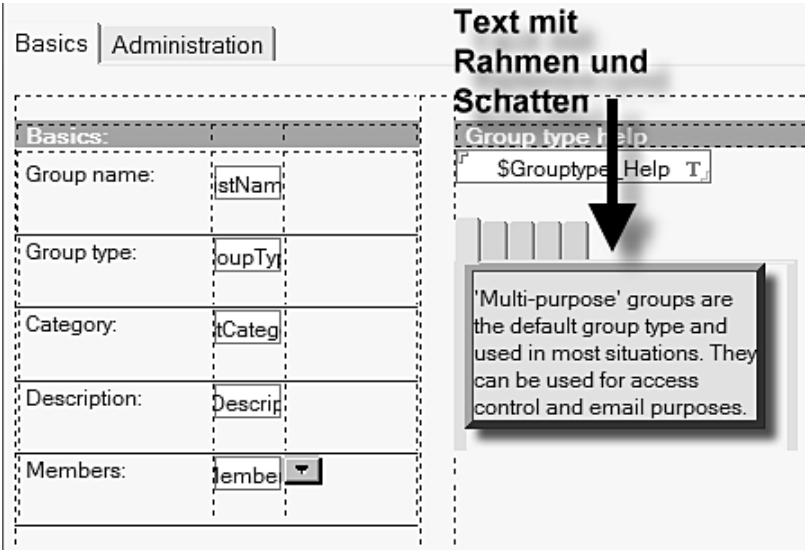
Berechneter Text

Im Bereich für Programmtexte kann jetzt ein Wert über eine Formel ermittelt werden. In der Beispielabbildung wird der Name des Benutzers ermittelt und in der Maske angezeigt. Die Texteigenschaften für den berechneten Text werden wie beim normalen Text über das Eigenschaftsfenster zugeordnet.



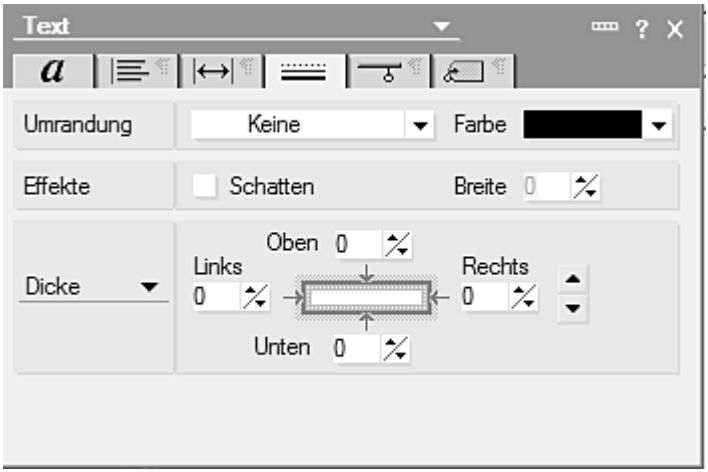
Eigenschaften des berechneten Textes

Eine interessante Gestaltungsmöglichkeit bietet der Designer für Texte durch Rahmen und Schatteneffekte. Der Text wird dann mit einem Rahmen versehen, dessen Eigenschaften einstellbar sind.



Text mit Rahmen und Schatten

Die Werkzeuge dazu finden sich in der Eigenschaftsbox zu Text im Register „Rahmen“.



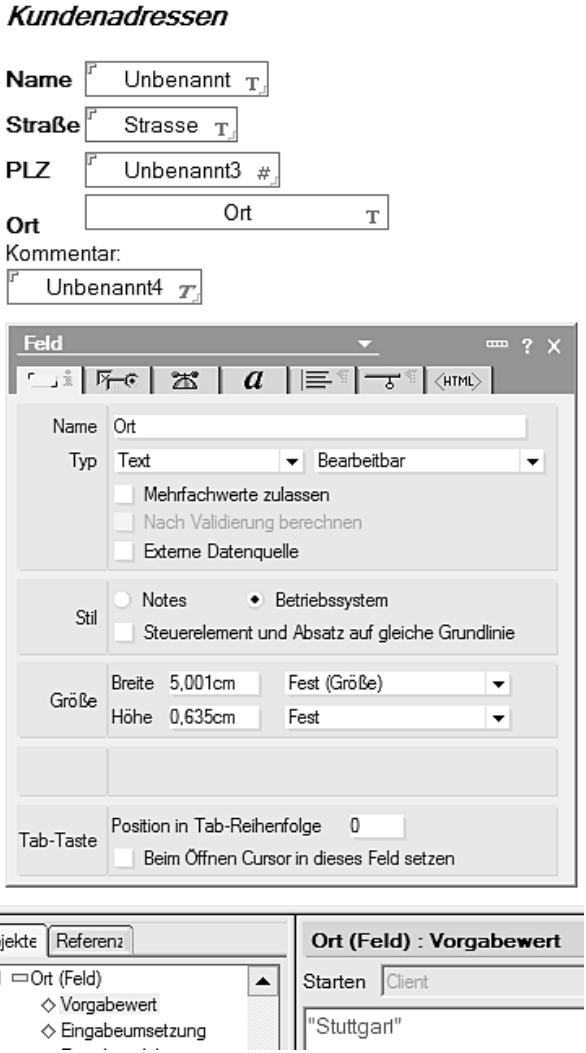
Texteigenschaften für Rahmen und Schatten

4.3.3.1.2 Felder

Felder werden in der Maske jeweils an der Cursor-Position eingefügt. Dazu findet sich im Menü „Erstellen“ die Position „Felder“. Der Designer fügt dann im Entwurfsbereich ein Feldsymbol ein. Dieses Feld ist mit einem Feldtyp fest verbunden, der standardmäßig auf „Text“ eingestellt ist. Durch ein Kennzeichen in dem Feldsymbol wird der Typ des Feldes anzeigt.

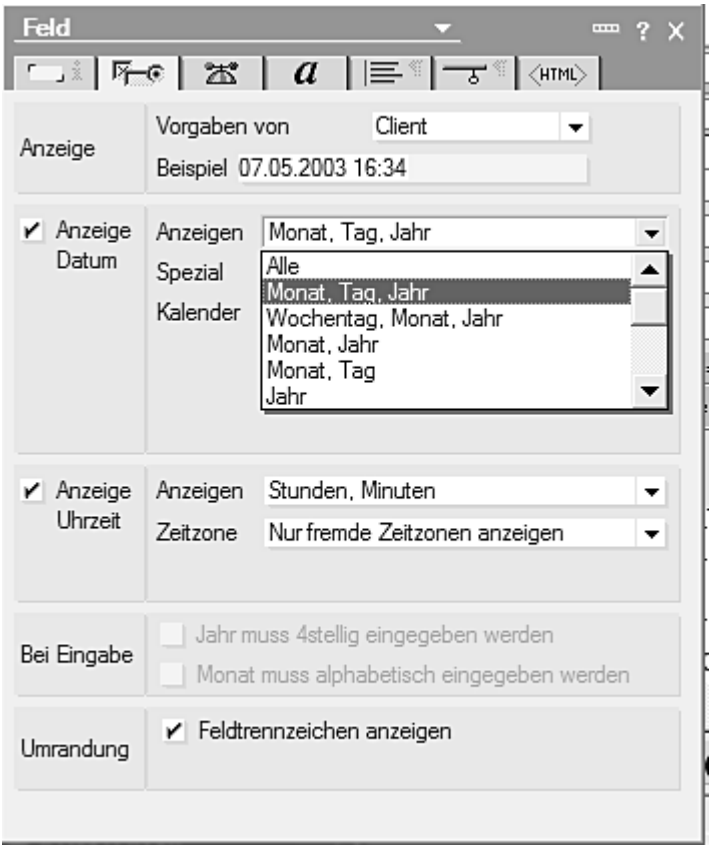
Neben den üblichen Feldtypen wie Text und Zahl kennt Notes eine ganze Reihe spezieller Typen. In den Masken werden folgende Feldtypen unterschieden:

- **Text**
  Ein Textfeld kann Text bis zu einer Länge von 15 KB Zeichen aufnehmen. Die Eigenschaften des Textes werden nicht vom Benutzer, sondern bereits vom Entwickler bestimmt.



Felder und Feldtypen

- **Zeit**
  Für Datum und/oder Uhrzeit verwendet Notes ein spezielles internes Format, das es erlaubt, problemlos mit Zeitzonen umzugehen**.** Die Anzeige dieser Felder kann in vielen verschiedenen Formaten erfolgen.



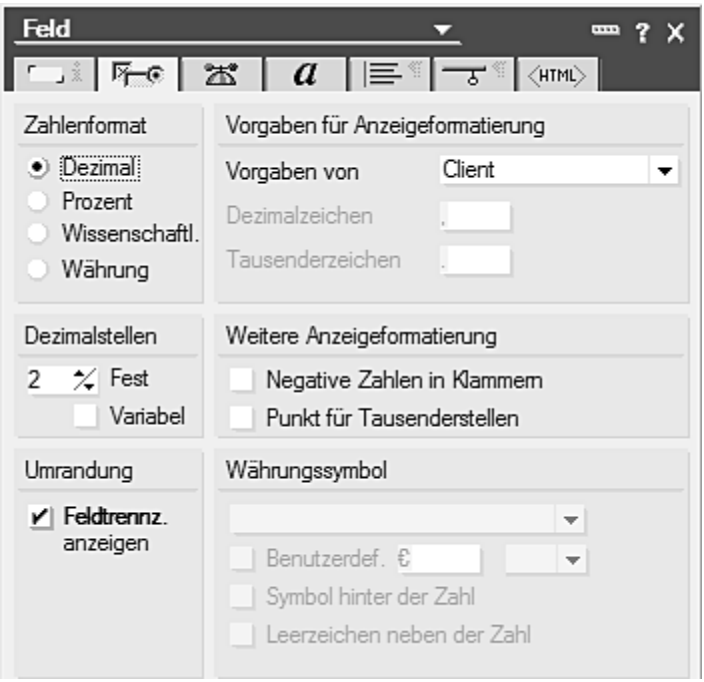
Darstellung von Datums- und Zeitfeldern

Da intern sowohl Datum als auch Uhrzeit gespeichert werden, wird die Anzeige im Register „Steuerung“ auf „Anzeige Datum“, „Anzeige Uhrzeit“ oder auf beide Werte gesetzt. Neben dem einfachen Erfassungsfeld können auch grafische Elemente zur Eingabe von Datum und Zeit eingesetzt werden.

Grafische Auswahlelemente für Datum und Zeit

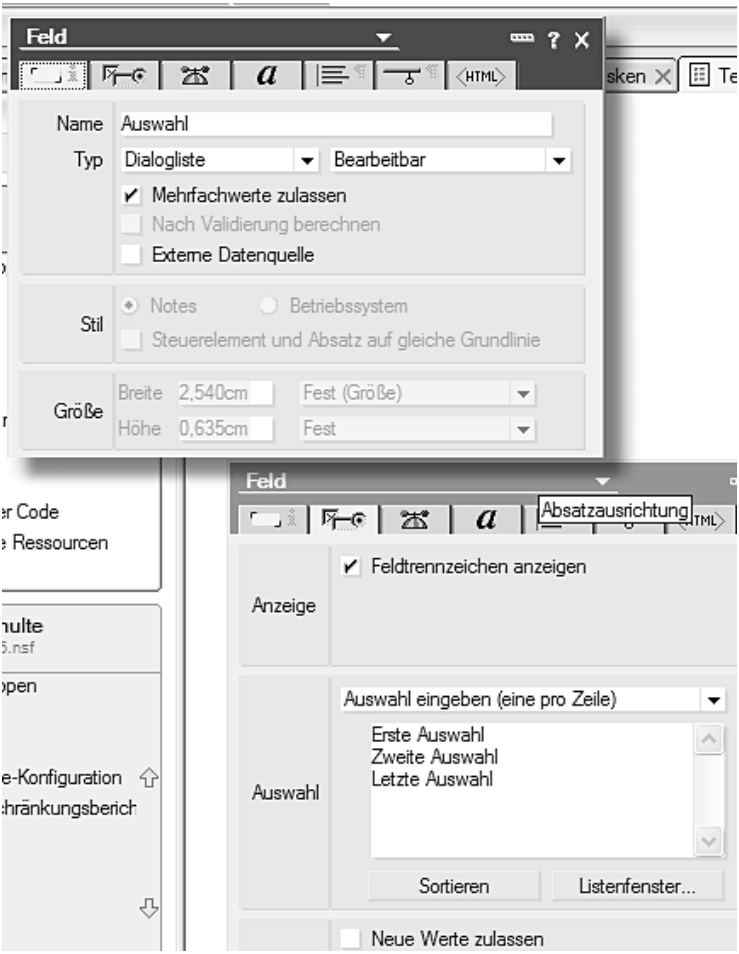
Dazu ist im Register „Allgemein“ im Bereich „Stil“ statt „Notes“ die Auswahl „Kalender- / Uhrzeit-Steuerelement“ zu treffen.

- **Zahl**
  Zahlen können als Dezimalzahlen, Prozentzahlen, Wissenschaftliche Zahlen und Währungszahlen verwaltet werden. Je nach Art der Zahl können weitere Angaben zur Formatierung gemacht werden.



Zahlenformate

- **Dialogliste**
  Die Dialogliste ist eines der Auswahlfelder, die der Designer anbietet. Es handelt sich dabei um ein Textfeld, das einen oder mehrere Werte aufnehmen kann. Soll das Feld mehrere Werte aufnehmen, so werden diese intern als Werteliste gespeichert.



Dialogliste

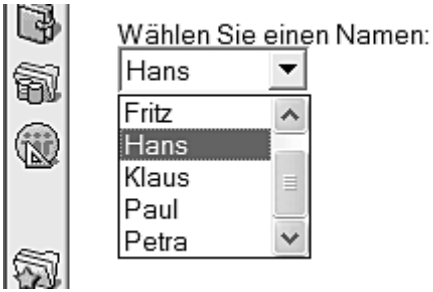
- **Kontrollkästchen**
  Diese Felder stellen sich als kleine Vierecke dar, die angehakt werden können. Ihr Wert kann also nur „1“ oder „0“ sein.

- **Optionsschaltfläche**
  Bei der Optionsschaltfläche handelt es sich um ein Auswahlfeld, bei der nur eine der gezeigten Möglichkeiten gekennzeichnet werden kann. Wird eine Auswahl angeklickt, werden automatisch alle anderen Möglichkeiten abgewählt.

Optionen und Kontrollkästchen

- **Listenfeld**
  Auswahlfenster, das in einer festen Größe angezeigt wird.

- **Kombinationsfeld**
  Kombinationsfelder öffnen sich beim Anklicken und zeigen eine Liste zur Auswahl an.

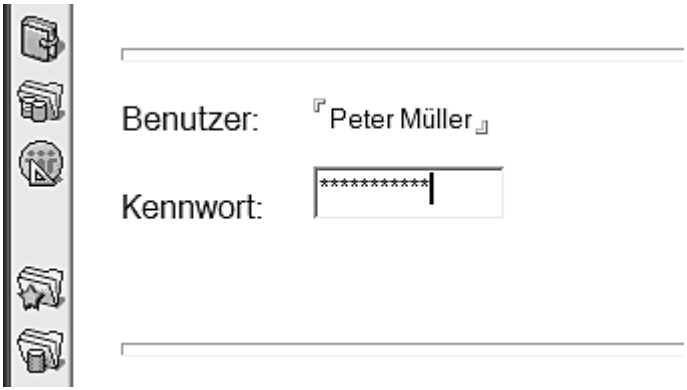


Kombinationsfeld

- **Richtext**
  Text im Richtext-Format. In solche Felder können gestaltete Texte, Bilder, Objekte und Datei-Anhänge gespeichert werden. Richtext-Felder können sehr groß werden. Die Objekte können beim Aufruf eines Dokumentes direkt gestartet werden.

- **Autoren, Leser, Namen**
  spezielle Namensfelder, die häufig zur Steuerung der Rechte an den Dokumenten verwendet werden. Die Namen werden im Notes-Format gespeichert und können mit Hilfe spezieller Formeln und LotusScript-Klassen in Programmen genutzt werden.

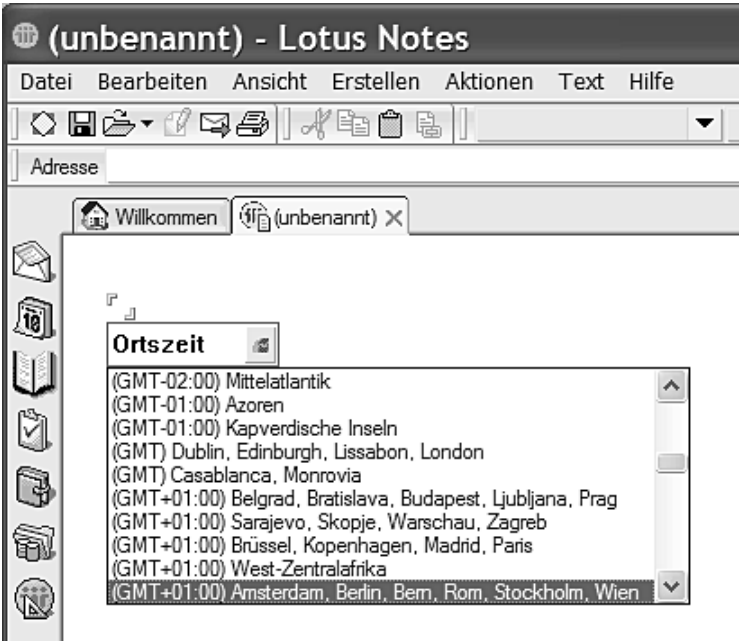
- **Kennwort**
  Bei der Eingabe in solche Felder wird der Text nicht direkt angezeigt. An seiner Stelle werden „*“-Zeichen in dem Feld gezeigt, wenn Zeichen erfasst werden.



Kennwort-Feld

- **Formel**
  Formelfelder werden zur Steuerung von Abonnements in der Datenbank „headlines.nsf“ verwendet.

- **Zeitzone (Release 6)**
  Auswahlfeld, in dem alle Zeitzonen der Welt zur Verwendung angeboten werden.



Auswahlfeld für Zeitzonen

- **Richtext Lite (Release 6)**
  Die Felder mit dem Typ „Richtext Lite“ unterscheiden sich von normalen Richtext-Feldern dadurch, dass die Objekttypen festgelegt werden können, die in das Feld eingefügt werden dürfen (z.B. nur Bilder oder nur Excel-Objekte). Bei dem Feld wird dann eine Schaltfläche angezeigt, über die z.B. ein Objekt, eine Datei oder ein Datumswähler gewählt werden kann. Daneben kann ein Kontextmenü geöffnet werden, über das festgelegt werden kann, welcher Daten- bzw. Objekttyp eingefügt werden soll. Außerdem kann definiert werden, ob der Benutzer das Feld löschen darf. Wird ein Hilfetext angegeben, so wird dieser in das Feld übernommen, wenn der Benutzer die Position „Hilfe zum OLE-Objekt“ auswählt.

Richtext-Lite-Feld

- **Farbe (Release 6)**
  Damit der Benutzer die Farbe von Gestaltungsfeldern ändern kann, wird ihm ein Farbwähler angeboten, der es erlaubt, die Farbe durch Anklicken zu übernehmen. Der Farbwähler ist mit einem Feld verbunden, das die ausgewählte Farbe übernimmt. Per Programm kann dieser Wert dann auf ein beliebiges Gestaltungselement (z.B. den Hintergrund der Maske) übernommen werden.

Farbwahl-Feld

Von besonderer Bedeutung ist das Richtext-Format. Es erlaubt die Erstellung von formatiertem Text, Text-Stilen und Tabellen. In Feldern dieses Typs können Objekte der unterschiedlichsten Art gespeichert werden. Damit ist es möglich, OLE-Programme zu verwenden sowie z.B. Word-Dokumente, Tabellen aus Lotus

1-2-3 oder Bilder aus Bildprogrammen in Domino-Dokumenten zu bearbeiten. Die Zugriffsmechanismen von Lotus Domino über Ansichten und Ordner werden mit den Funktionen einzelner Programme kombiniert. Einziger Nachteil dieser Vorgehensweise ist der große Bedarf an Systemressourcen (Speicher und Prozessor).

Felder können bei der Anzeige und/oder der Bearbeitung von Dokumenten verborgen werden. Solche Felder dienen dazu, feste oder berechnete Werte vorzuhalten, die bei den Berechnungen anderer Felder oder für Ansichten benötigt werden.

Felder haben neben der reinen Erfassungsfunktion auch die Möglichkeit, Werte über Programm-Module zu berechnen. Dabei gibt es vier Varianten

- **bearbeitbar**
  Hier kann als Vorgabewert eine Berechnungsformel hinterlegt werden.

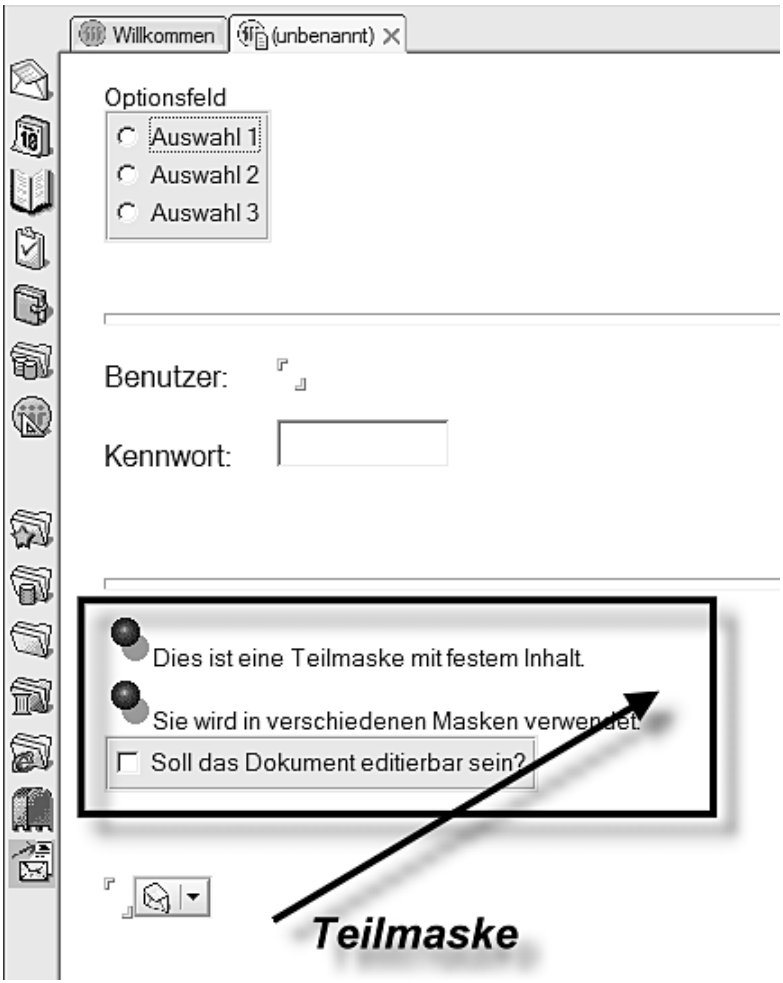
- **berechnet**
  Bei berechneten Feldern ist keine Bearbeitung möglich, der Inhalt wird über eine Formel berechnet und in der Datenbank gespeichert.

- **bearbeitet zur Anzeige**
  Der berechnete Wert wird nicht gespeichert, sondern jedes Mal neu ermittelt, wenn eine Neuberechnung der Maskenfelder erfolgt.

- **berechnet beim Anlegen**
  Der Feldinhalt wird nur bei der Erstellung eines neuen Dokumentes ermittelt (kann aber per Programm verändert werden).

4.3.3.2 Teilmasken

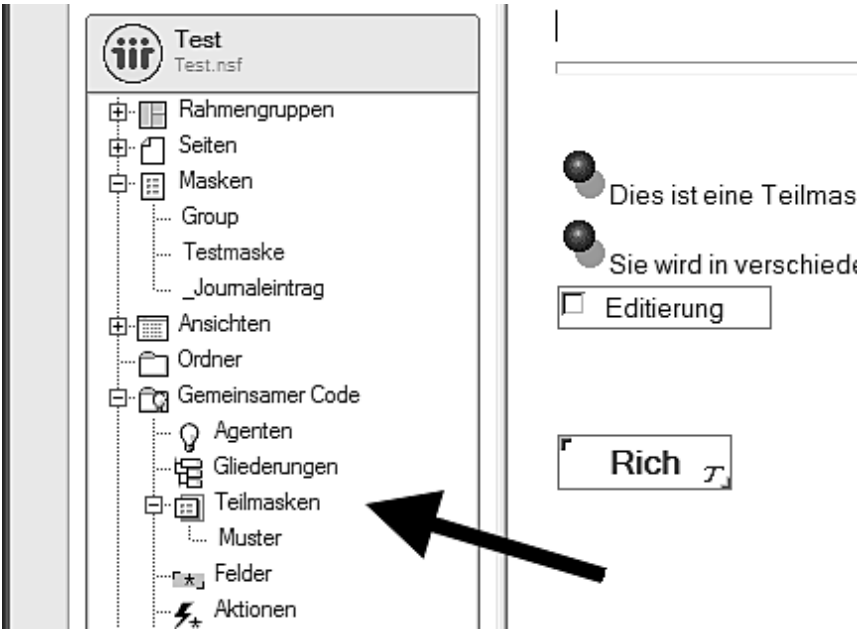
Eine weitere Technik zur mehrfachen Verwendung von Programmteilen stellt der Domino-Designer mit den Teilmasken zur Verfügung.



Teilmaske

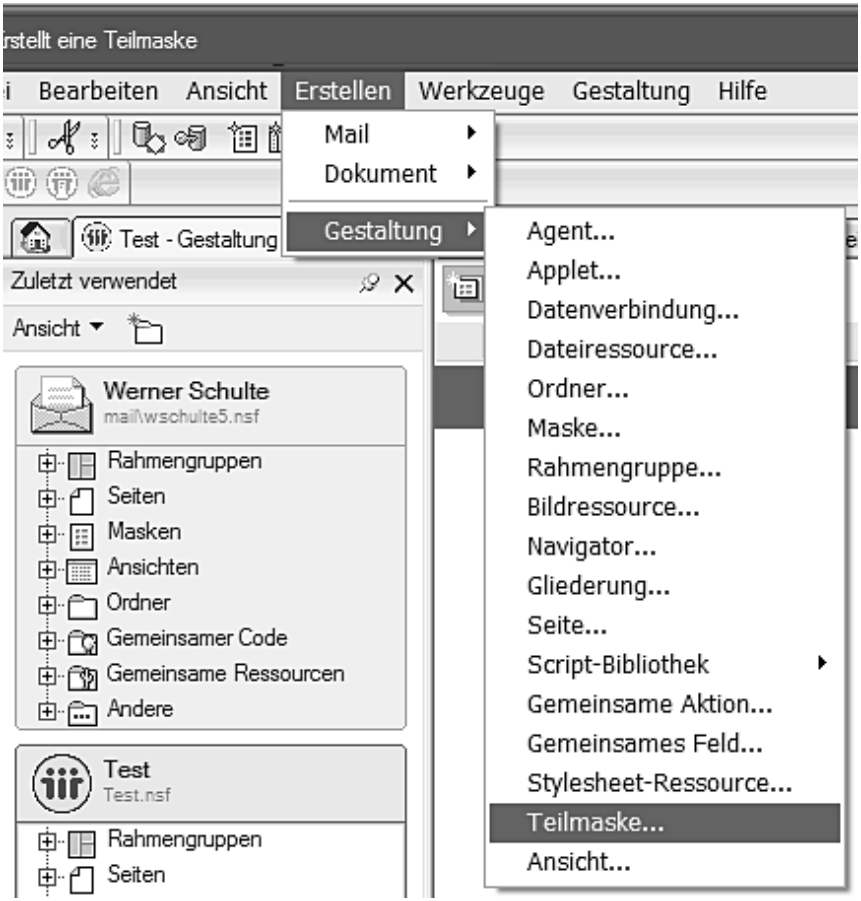
Eine Teilmaske enthält einen Maskenausschnitt und wird in Masken eingefügt. Alle Änderungen, die man an einer Teilmaske durchführt, werden automatisch in alle Masken übernommen, die diese Teilmaske enthalten. Eine Maske kann mehrere Teilmasken enthalten. Damit lassen sich Bausteine zusammenstellen, die viel zu einem einheitlichen Erscheinungsbild einer Anwendung beitragen.

Teilmasken werden ab Release 6 unter der Rubrik „Gemeinsamer Code“ und dort unter „Teilmasken“ verwaltet.



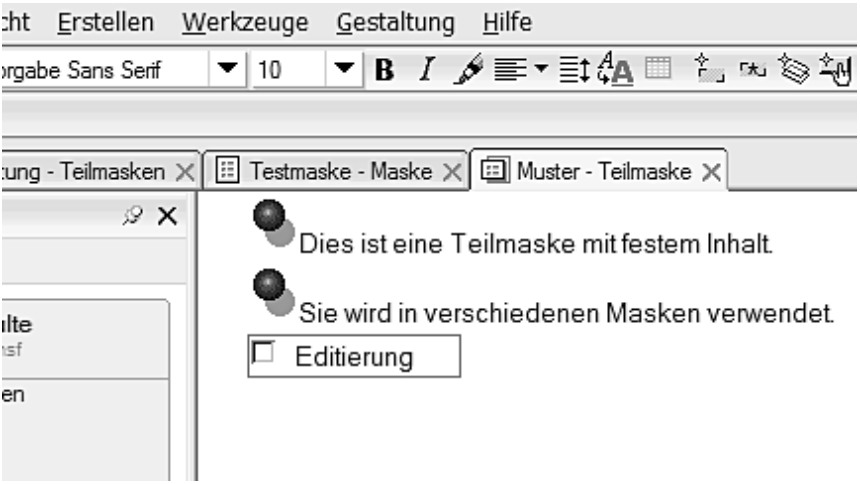
Teilmaskenverwaltung

Um eine neue Teilmaske zu erstellen, klickt man einfach die Schaltfläche „Neue Teilmaske“ an oder man wählt im Menü die Position „Erstellen / Gestaltung / Teilmaske“. In beiden Fällen wird eine leere Teilmaske zur Bearbeitung gewählt.



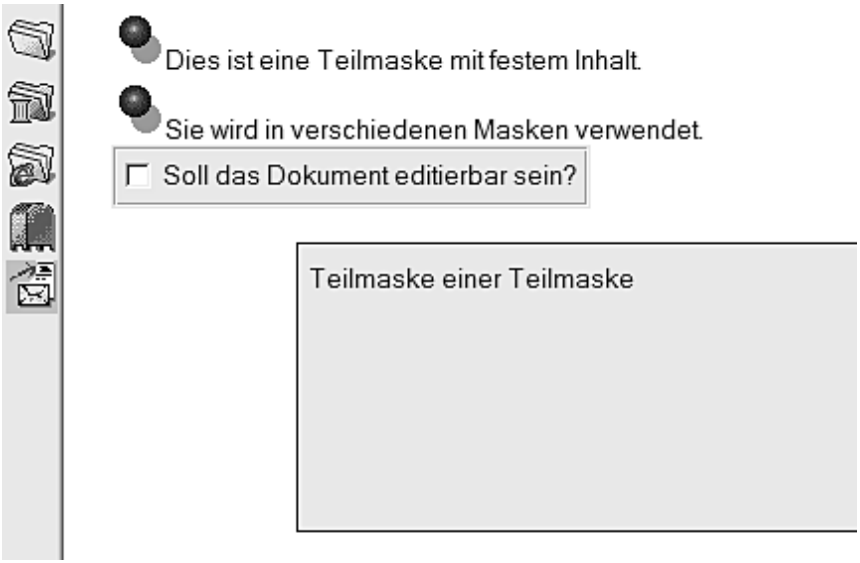
Teilmaske erstellen

In der Teilmaske können alle Elemente eingearbeitet werden, die man in einer Maske benötigt, also Felder, Texte, Aktionen etc. Diese Form der Aufteilung einer Maske in Bausteine wird häufig dazu benutzt, einen einheitlichen Kopf- oder Fußbereich von Masken zu gestalten. Es sind aber auch viele andere Einsatzmöglichkeiten denkbar, die wichtige Funktionen als Teilmaske zusammenfassen und in Masken verwenden.



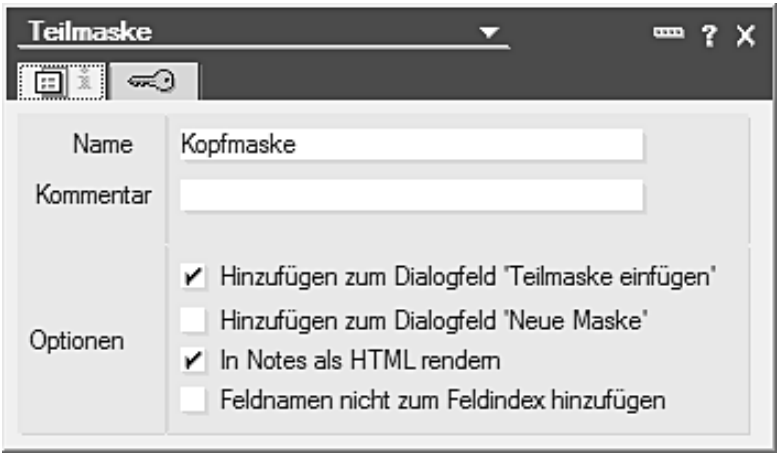
Teilmaske bearbeiten

Sogar Teilmasken können wieder in eine Teilmaske eingefügt werden.



Teilmaske mit Teilmaske

Teilmasken sind von ihren Eigenschaften und Ereignissen her eingeschränkt, da sie in Masken eingebunden werden und deren Eigenschaften und Ereignisse übernehmen.



Eigenschaften der Teilmaske

Da eine Teilmaske Elemente enthalten kann, die von Maske zu Maske abweichen, gibt es eine ganze Reihe von Maßnahmen, die es erlauben, Teilmasken dynamisch zu gestalten, also sie mit Inhalten zu versehen, die sich an die Zielmaske anpassen. Wie dabei vorgegangen werden kann, soll hier an einem Beispiel gezeigt werden.

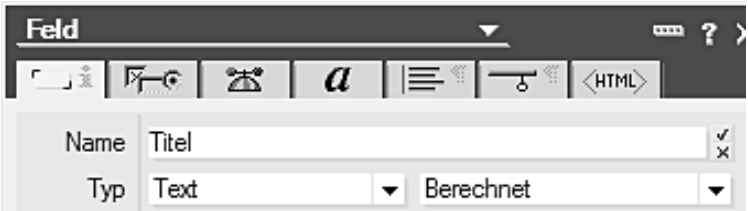
Zunächst wird eine Teilmaske erstellt, die den Namen Kopfmaske tragen soll. In dieser Teilmaske wird ein Titel, ein Datum und ein Logo angezeigt. Titel und Datum sollen sich jeweils automatisch aktualisieren bzw. an die Maskenumgebung anpassen.

❶❷❸ Ausführungsschritte:

1. Unter der Rubrik „Gemeinsamer Code“ die Position “Teilmasken” anklicken.

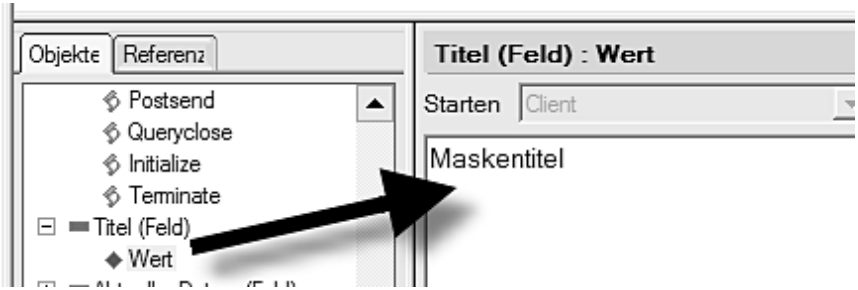
2. Die Schaltfläche „Neue Teilmaske“ drücken.

3. Textfeld für den Titel erstellen (Name: Titel, Typ: Text, berechnet).



Titelfeld

4. Formel zur Berechnung des Feldinhaltes ergänzen: Im Programmbereich wird im Ereignis „Wert“ des Titelfeldes eine Formel eingetragen. In diesem Fall ist das der Name des Feldes, in dem in der Hauptmaske der Titel steht. Dieses Feld heißt „Maskentitel“.



Wert für das Titelfeld

5. Feld für das Datum erstellen (Name: Aktuelles Datum, Typ: Datum/Zeit, Berechnet zur Anzeige).

Datumsfeld

6. Formel zur Berechnung des Datums einfügen: Im Ereignis „Wert“ des Datumsfeldes wird die Formel „@Today“ eingetragen.

7. Das Datumsfeld – z. B. über die Tabulatortaste – vom Titelfeld absetzen.

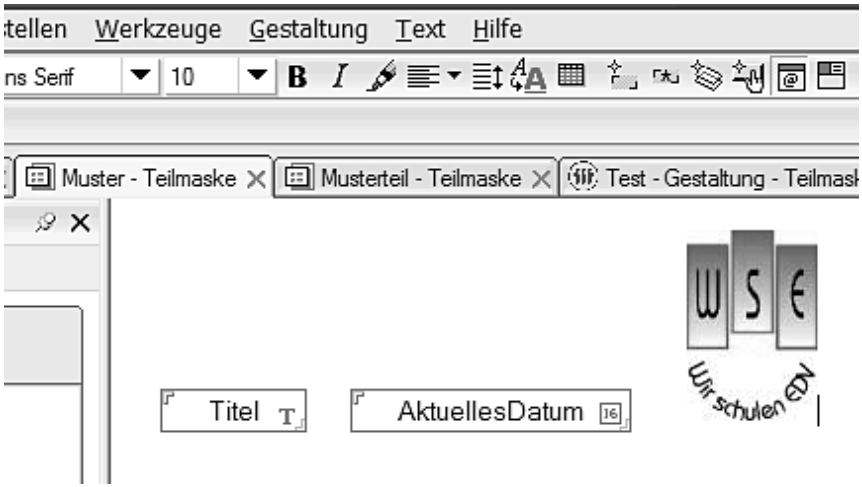
8. Logo einfügen (Im Menü „Erstellen“ die Position „Bildressource“ verwenden. Ist das Logo nicht in einer Datenbank verfügbar, dann über „Neu“ eine Ressource erstellen).

Neue Bildressource

9. Eventuell weitere Bild- oder Gestaltungselemente einfügen.

10. Teilmaske unter dem Namen „Kopfmaske“ speichern.

Die fertige Teilmaske wird jetzt in der Liste der Teilmasken aufgeführt und kann in Masken verwendet werden.



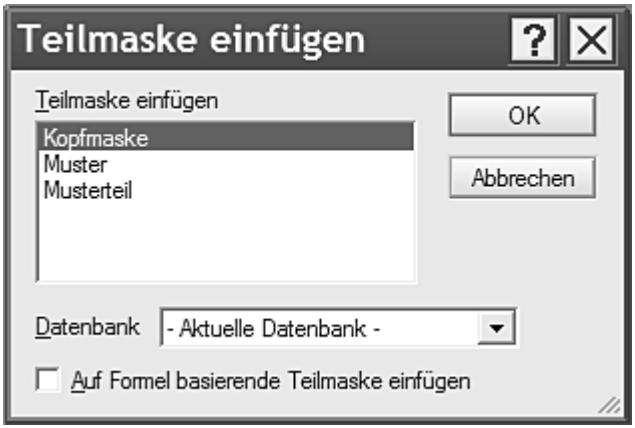
Fertige Teilmaske

Zur Verwendung der Teilmaske erstellt man eine neue leere Maske und fügt die Teilmaske ein. Damit der Titel im Titelfeld des Kopfes erscheint, wird in der Maske ein Feld mit dem Namen der Maske eingefügt.

❶❷❸ Ausführungsschritte:

1. Im Menü „Erstellen“ die Position „Gestaltung / Maske“ ausführen.

2. In der Maske den Cursor an den Anfang setzen und über „Erstellen / Ressource / Teilmaske einfügen“ das Dialogfeld zum Einfügen der Teilmasken öffnen.



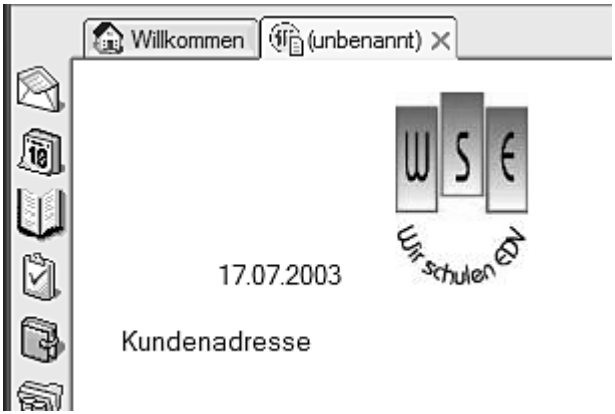
Teilmaske einfügen

3. Die Teilmaske „Kopfmaske“ auswählen und mit „OK“ einfügen.

4. Die Teilmaske wird jetzt in der neuen Maske angezeigt.

5. Ein Feld mit dem Namen „Maskentitel“ erstellen (Name: Maskentitel, Typ: Text, Berechnet, Wert: „Kundenadresse“).

6. Maske mit dem Namen „Kundenadresse“ speichern und in der Notesvorschau anzeigen.



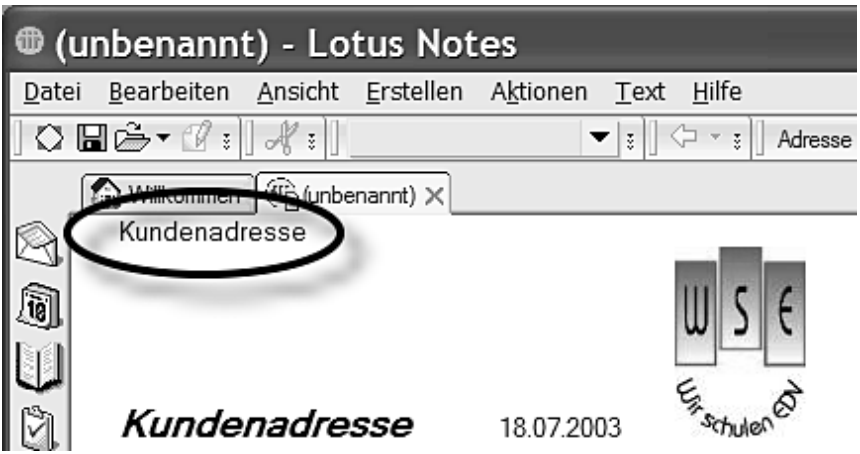
Maskenvorschau

Das Titelfeld der Teilmaske wird offensichtlich noch nicht gefüllt.

7. Das Feld „Maskentitel“ muss vor der Teilmaske platziert werden, damit der Feldinhalt berechnet wird, bevor die Teilmaske erstellt wird.

Feld vor der Teilmaske einfügen

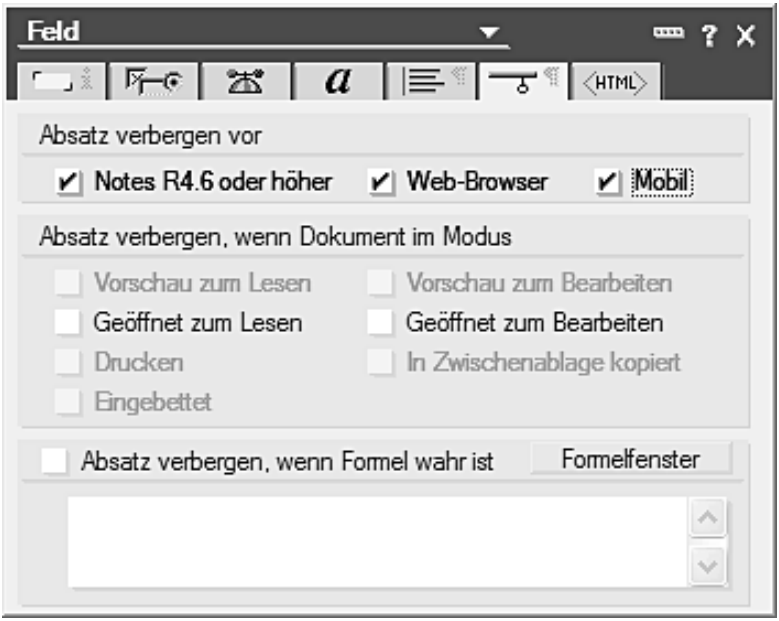
8. Maske speichern und testen.



Fertige Maske mit Titel Datum und Logo

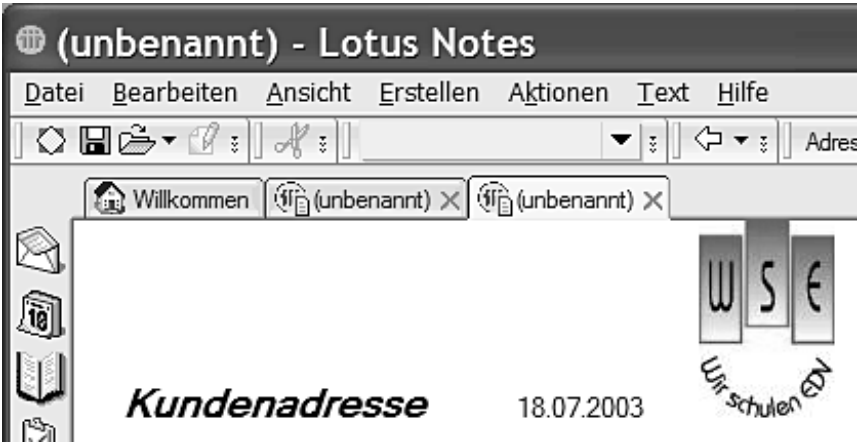
Am oberen Rand der Maske wird jetzt das Feld „Maskentitel“ angezeigt. Im Feld „Titel“ der Teilmaske wird dieser Text übernommen und gezeigt. Damit ist die Teilmaske universell einsetzbar, wenn in der eigentlichen Maske ein Feld „Maskentitel“ den anzuzeigenden Text enthält.

9. Die Anzeige des Feldes „Maskentitel“ sollte noch unterdrückt werden. Dazu wird in den Eigenschaften dieses Feldes im Register „Verbergen-wenn“ im Bereich „Absatz verbergen vor“ alles angehakt.



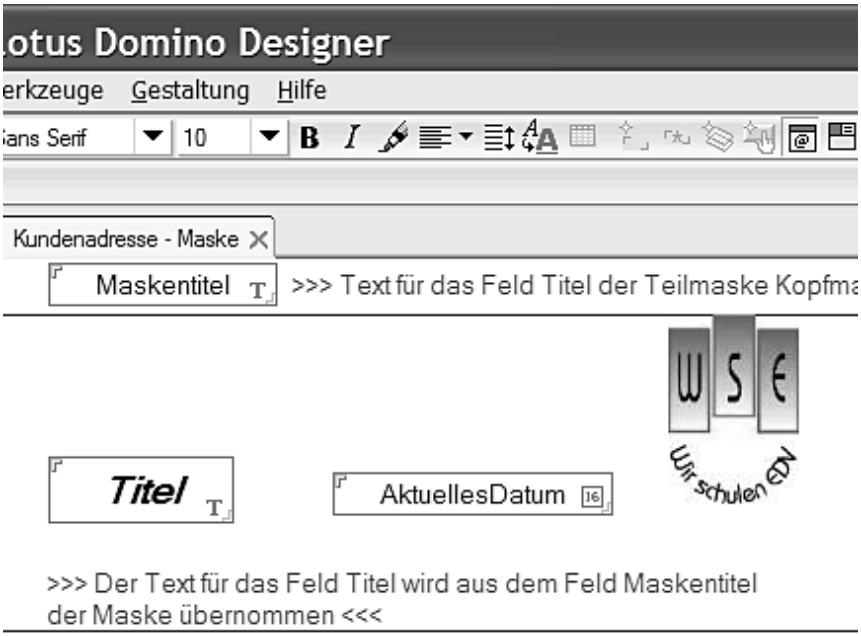
Feld Maskentitel verbergen

Dadurch wird das Feld im Notes-Client und auch im Web-Browser nicht mehr angezeigt.



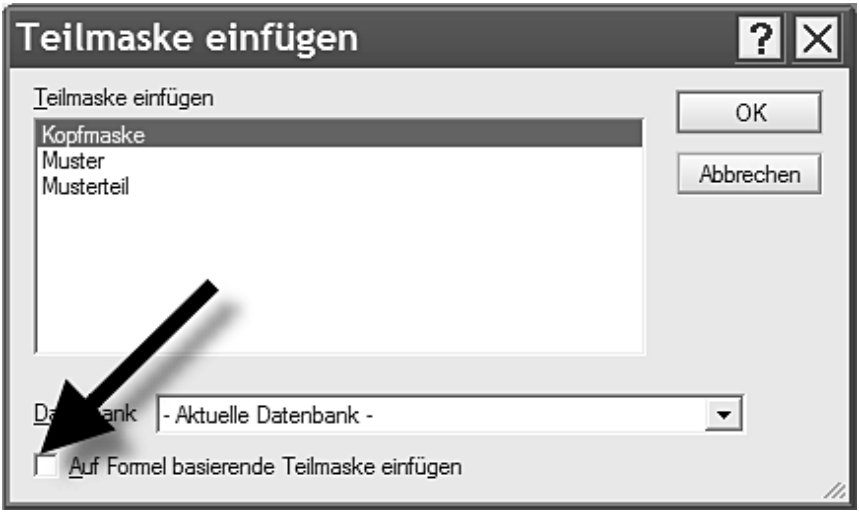
Maskenkopf ohne das Feld „Maskentitel“

10. Damit der Entwickler später noch alle Informationen zu der Funktion und den Abhängigkeiten der Teilmaske hat, empfiehlt es sich, die Maske und die Teilmaske mit einem Kommentar zu versehen. Der Kommentar wird ebenfalls über seine Eigenschaften als verborgen gekennzeichnet.



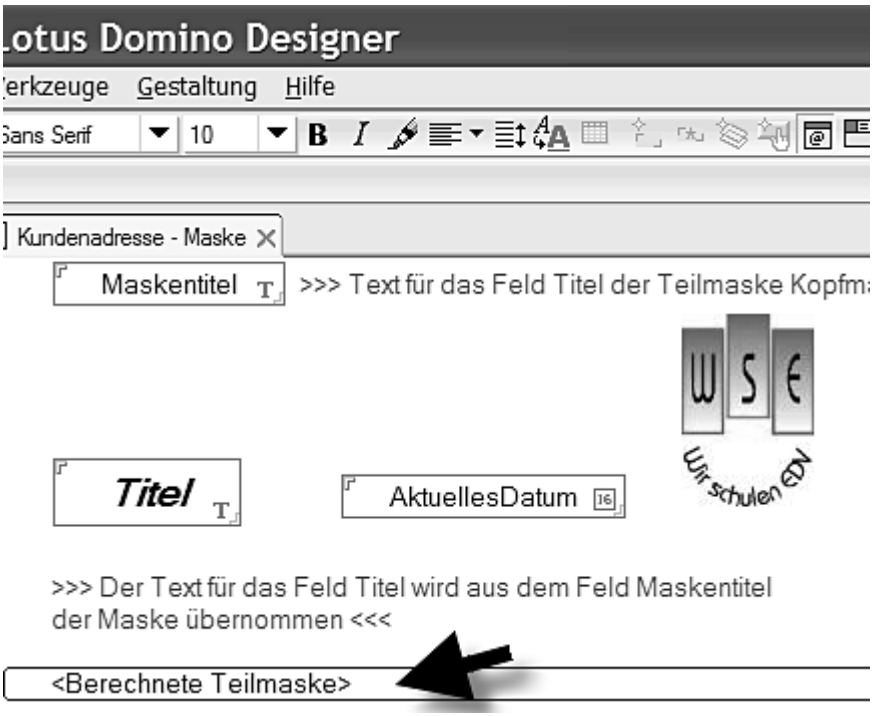
Kommentare in Maske und Teilmaske

Eine weitere Möglichkeit, Teilmasken dynamisch einzusetzen, stellt die berechnete Teilmaske dar. Diese Variante bezieht sich nicht auf die Teilmaske selbst, sondern auf ihren Einfügemodus in die Hauptmaske. Zur Laufzeit, also dann, wenn die Maske am Bildschirm angezeigt werden soll, wird anhand einer Formel geprüft, welche Teilmaske an einer bestimmten Stelle erscheinen soll. Damit kann sowohl auf die Maske als auch auf Feldinhalte und Benutzertypen reagiert werden.



Berechnete Teilmaske einfügen

In der Maske wird die berechnete Teilmaske als Zeile angezeigt, die den Text „<Berechnete Teilmaske>“ trägt.



Anzeige einer berechneten Teilmaske

Bei der Darstellung der Maske für den Benutzer zeigt der Notes-Client die Teilmaske in Abhängigkeit von der Formel an, die bei der Maske hinterlegt ist.

❶❷❸ Ausführungsschritte:

Bevor die berechnete Teilmaske zur Anwendung kommt, sollten die Teilmaske oder die alternativen Teilmasken bereits erstellt sein.

1. Über das Menü „Erstellen / Ressourcen / Teilmaske einfügen“ den dazu gehörenden Dialog starten.

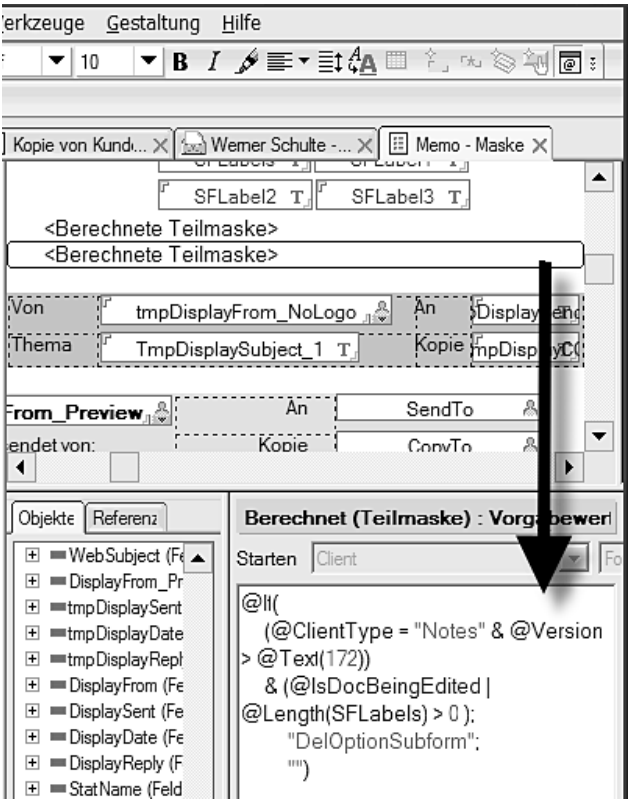
2. Das Kontrollfeld „Auf Formel basierende Teilmaske einfügen“ aktivieren”.

3. Im Ereignis „Vorgabewert“ der Teilmaske die Auswahlformel definieren.

Formel für die Teilmaske

4. Die Maske auf ihre Funktion hin testen.

Eine wichtige Anwendung ist die Darstellung von Alternativen im Internet und im Notes-Client. Dabei wird über eine Formel geklärt, welcher Modus aktiv ist und dann die entsprechende Teilmaske gezeigt.



Berechnete Teilmasken für die Anzeige im Web

Natürlich ist es auch möglich, die Anzeige von berechneten Teilmasken über Feldinhalte zu steuern. Dann gilt wieder, dass das Feld mit dem steuernden Inhalt vor der Teilmaske in der Maske platziert wird.

4.3.3.3 Tabellen

Die Anordnung von Elementen in einer Maske ist zunächst rein textorientiert. Der Text wird in Zeilen erstellt und durch Zeilenschaltungen nach unten erweitert. Die anderen Elemente wie Felder, Bilder, Teilmasken usw. werden jeweils in diese Zeilenstruktur eingefügt. Der Domino-Designer verfügt inzwischen über Elemente wie Layout-Regionen und Ebenen, die es erlauben, die Elemente in einer Maske wesentlich eleganter zu positionieren. Aber auch heute wird diese Aufgabe meist durch den Einsatz von Tabellen gelöst.

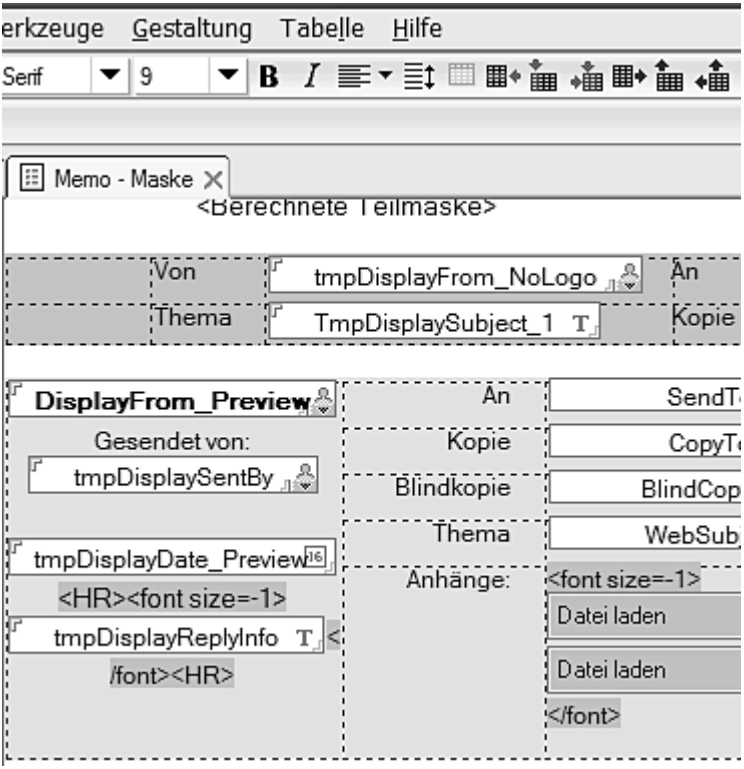


Tabelle in der Memo-Maske

Über die Einteilung und die Eigenschaften einer Tabelle werden Texte, Felder und andere Gestaltungselemente so angeordnet, dass die Maske möglichst genau mit den funktionellen und gestalterischen Wünschen des Entwicklers übereinstimmt.

Mit Release 5 wurde die Tabelle um eine ganze Reihe neuer Möglichkeiten erweitert. Die Tabelle ist dadurch zu einem attraktiven Gestaltungselement geworden, das weit über die reine Positionierung hinaus eingesetzt werden kann. In Release 6 wurden weitere Funktionen ergänzt.

Neue Tabelle erstellen

Es stehen jetzt fünf verschiedene Tabellentypen zur Verfügung, die zu unterschiedlichen Zwecken eingesetzt werden können:

- **Einfache Tabelle**
  Tabelle mit Zeilen und Spalten.

- **Tabelle mit Registern**
  Für jede Zeile wird eine Registerseite angezeigt.

- **Animierte Tabelle**
  Zeigt jede Zeile nur für eine festgelegte Zeit an (Banner-Effekt).

- **Tabelle mit Titelzeilen**
  Bei dieser Tabellenart hat jede Zeile einen Titel zum Anklicken.

- **Programmierte Tabelle**
  Die Anzeige der Tabellenzeilen wird über ein Feld gesteuert, das per Programm mit dem entsprechenden Namen gefüllt wird.

Zum Einfügen einer Tabelle wird im Menü „Erstellen“ die Position „Tabelle“ ausgewählt. Es erscheint ein Dialogfenster, in dem neben dem Tabellentyp die Anzahl der Zeilen und Spalten und die Tabellenbreite angegeben werden. Der Designer fügt dann eine entsprechende Tabelle an der Cursor-Position ein.

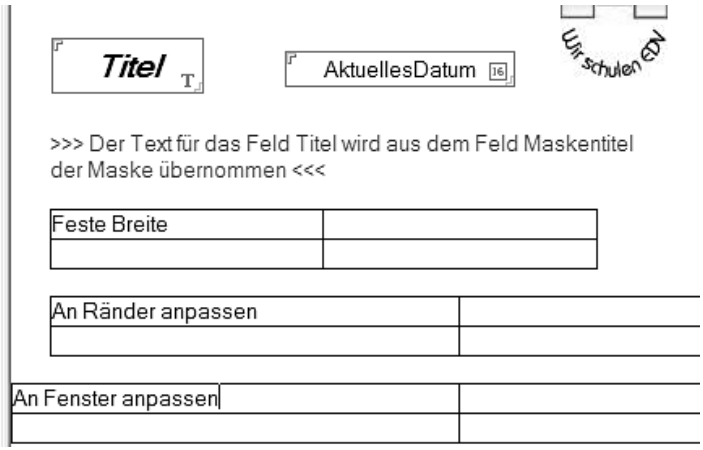
❶❷❸ Ausführungsschritte:

1. Cursor an die Stelle setzen, an der die Tabelle eingefügt werden soll.

2. Im Menü „Erstellen“ die Position “Tabelle” auswählen.

3. Anzahl der Zeilen und Spalten angeben.

4. Tabellenbreite festlegen.
   Mögliche Werte sind hier:
   – „An Ränder anpassen“ (flexible Breite zwischen den festgelegten Rändern der Seite oder des Feldes, in das die Tabelle eingebettet wird.)
   – „An Fenster anpassen” (flexible Breite in dem vom Fenster vorgegebenen Rahmen) und „Feste Breite“ (Konstante Breite).

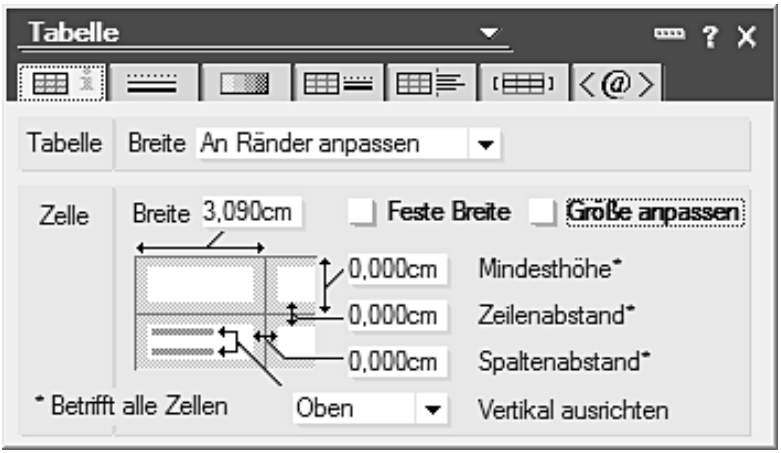


Tabellenbreite festlegen

5. Tabellentyp wählen.
   Zunächst den Typ „Einfache Tabelle“ auswählen.

6. Dialog abschließen.

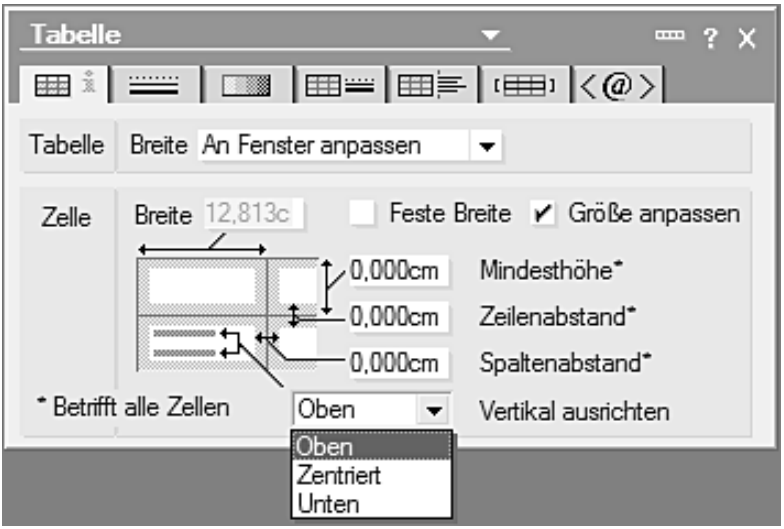
Ist die Tabelle erstellt, können jederzeit Änderungen an ihren Eigenschaften vorgenommen werden. Die Eigenschaften der Tabelle sind unterteilt in verschiedene Bereiche, die über das Register angezeigt werden können. Das erste Register ist für das Tabellenlayout zuständig und erlaubt es, die Eigenschaften zu bearbeiten und zu verändern, die bei der Erstellung der Tabelle angegeben wurden.



Tabelleneigenschaften

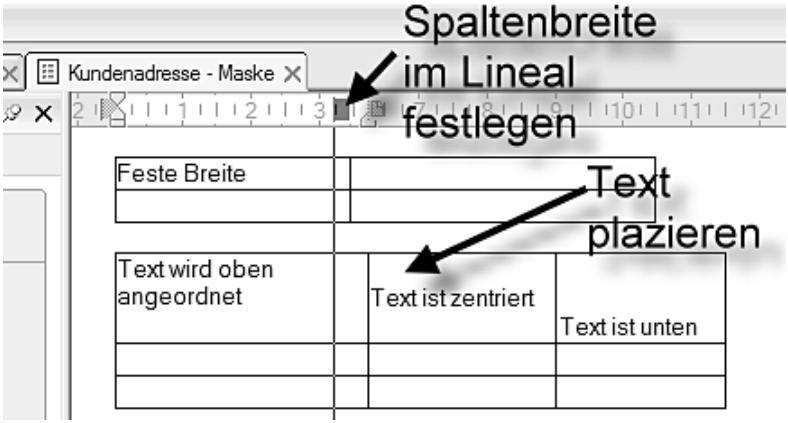
Zunächst wird wieder definiert, ob die Tabelle eine feste Breite hat oder ob sie an die Breite der Seite oder des Fensters gekoppelt ist. Danach können Werte für die Zellen festgelegt werden. Die Breite der Spalten bezieht sich immer auf die Spalte, in die der Cursor positioniert wurde. Wenn die Tabelle sich an die Ränder oder das Fenster passt, ist die Tabellenbreite vorgegeben. Da in diesem Fall vorgegeben ist, welchen Wert die Summe aller Spalten ergeben muss, kommt es auch zu Veränderungen anderer Spalten, wenn eine Spalte verändert wird. Soll z. B. die Breite einer Spalte 4 cm betragen, so wird dieser Wert im Feld „Breite“ eingetragen und das Kontrollfeld „Feste Breite“ angehakt. Das Kontrollfeld „Größe anpassen“ bewirkt, dass sich die Breite der Spalte nach der Breite des eingegebenen Textes richtet. Da sich das nicht mit dem Wert „Feste Breite“ verträgt, wird dieses Feld automatisch angepasst, wenn das Feld „Größe anpassen“ angeklickt wird.

Die Werte für Mindesthöhe, Zeilenabstand und Spaltenabstand gelten für alle Zellen der Tabelle. Damit kann erreicht werden, dass die Tabelle ein einheitliches Aussehen erhält, das unabhängig vom eingegebenen Text ist. Werden die Werte nicht verwendet, hängen Sie von der verwendeten Schriftgröße, den Leerzeichen und Zeilenschaltungen ab. Die Anordnung der Texte innerhalb der Zellen kann für jede Zelle einzeln festgelegt werden.



Textausrichtung in Zellen

Mögliche Werte sind: „Oben“, „Zentriert“ und „Unten“.

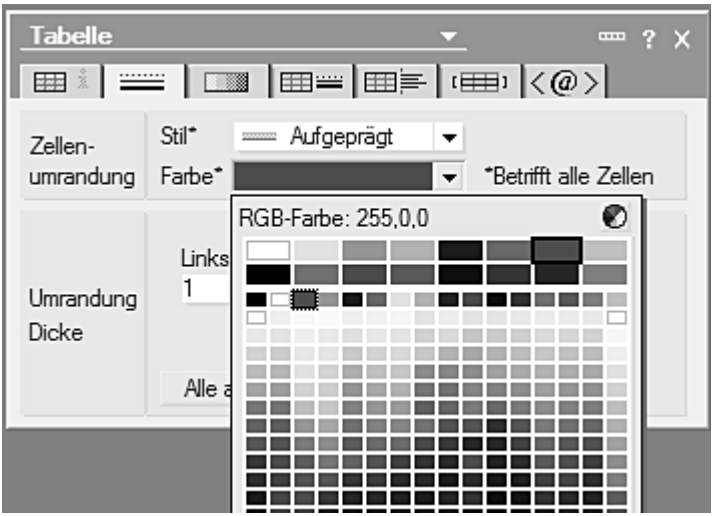


Zellen-Eigenschaften

Die Spaltenbreite kann auch dadurch verändert werden, dass das Lineal eingeblendet und die Randsymbole verschoben werden.
Im Register „Zellenumrandung“ werden die Eigenschaften für Farbe, Stil und Stärke der Linien definiert, die die Zellen umgeben.

Zellen-Umrandung

Stil und Farbe können nur für die ganze Tabelle eingestellt werden.



Farbe der Zellen-Umrandung

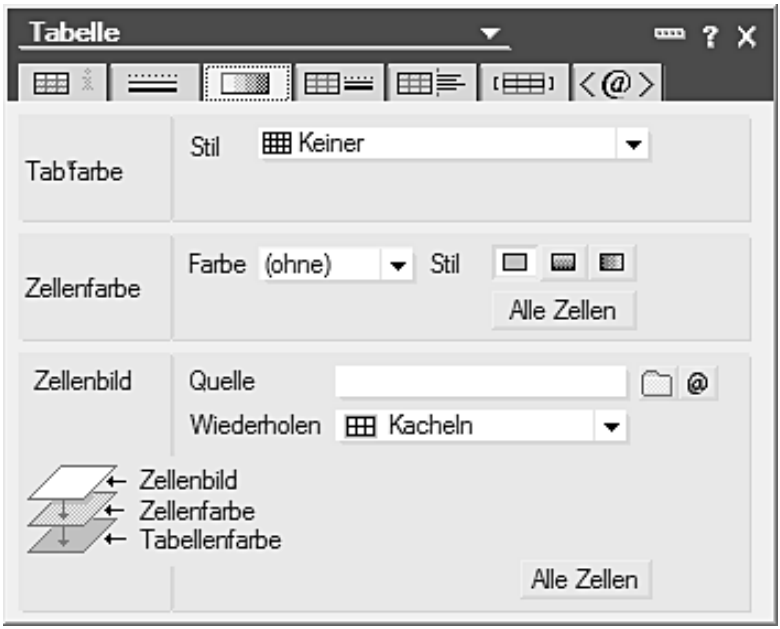
Die Stärke des Randes kann bei den Zellen an jeder Seite unterschiedlich sein.

Text wird oben angeordnet	Text ist zentriert	Text ist unten
	Unterschiedliche Ränder bei einzelnen Zellen	

Zellen mit unterschiedlichem Rand

Über die Schaltflächen „Alle auf 0“ und „Alle auf 1“ können wieder einheitliche Randstärken für alle Zellen eingestellt werden.

Das Register „Tabellenfarben“ erlaubt es, der Tabelle und ihren Zellen Farben und Bilder zuzuordnen.



Farben und Bilder in der Tabelle

Auf der untersten Ebene dieser Darstellungshierarchie werden Farbe und Stil für die Tabelle angegeben. Sie gelten für alle Zellen, solange dort nicht andere Vorgaben gemacht werden. Da Tabellen für ganz unterschiedliche Zwecke (Masken, Berichte, Textformatierung) eingesetzt werden, bietet das Farbregister eine Stil-Sammlung, in der unterschiedliche Farbkombinationen ausgewählt werden können.

Farbstile

Wird kein Stil gewählt, so wird die Tabelle farblos, also weiß dargestellt. Der Wert „Einfarbig“ weist die Farbe einheitlich der ganzen Tabelle zu.

Farben für Farbstil „Links“

Ist ein Stil definiert, so kann eine Farbe für den hervorgehobenen und eine für den Rest eingestellt werden. Die Stile orientieren sich vorwiegend an den Bedürfnissen von Auswertungen und Berichten. Die typischen Tabellenauswertungen werden durch Kopf- und Seitenfarben bzw. durch alternierende Zeilen- oder Spaltenfarben lesbarer gestaltet.

Name		
Adresse		
Bemerkung		

Tabelle mit Farbstil „Links“

Die Farben der Zellen können für jede einzelne Zelle festgelegt werden und überschreiben die Farbe der Grundtabelle.

Kombination aus Tabellen- und Zellenfarbe

Interessant ist auch die Möglichkeit, der Farbe in einer Zelle einen Farbverlauf zuzuordnen. Der Verlauf kann von oben nach unten oder zur Seite hin angeordnet sein.

Name		
Adresse	Zelle mit eigener Farbe	Zelle mit Verlauf
Bemerkung		

Zelle mit Farbverlauf

Die Farben können in den Feldern „Farbe“ und „nach“ eingestellt werden. Die Schaltfläche „Alle Zellen“ erlaubt es, die Tabellenfarbe für alle Zellen auf die gleiche Farbe bzw. den gleichen Stil zu setzen. Normalerweise gilt die Farbdefinition immer für die Zelle, in der der Cursor gerade steht.

Neben einer Farbe kann auch ein Bild als Hintergrund für eine Zelle eingesetzt werden.

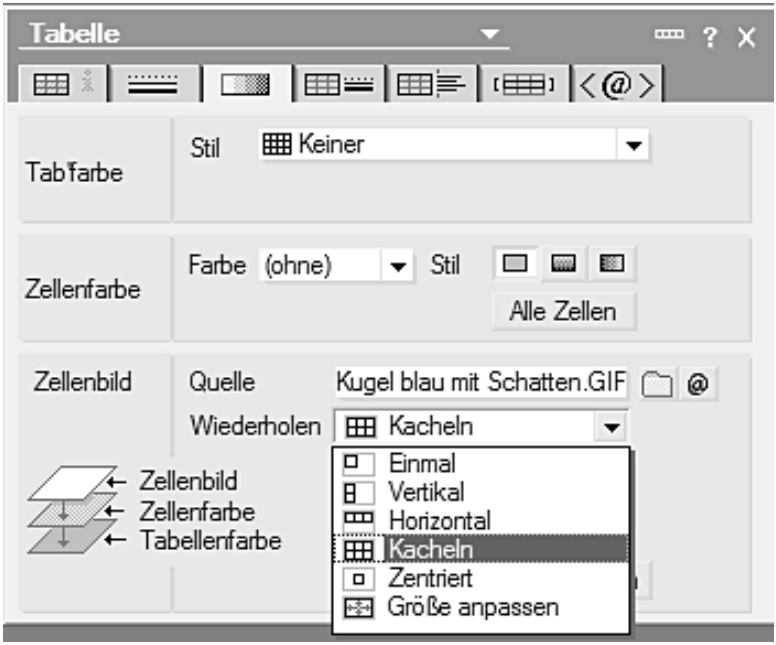
Name		
Adresse	Zelle mit eigener Farbe	Zelle mit Verlauf
Bemerkung		

Zelle mit Bild

Als Quelle für das Hintergrundbild einer Zelle dienen die Bildressourcen der aktuellen oder einer anderen Datenbank. Ist keine passende Bildressource vorhanden, so kann sie angelegt werden. Das Bild kann dann auf unterschiedliche Arten in der Zelle angeordnet werden:

- **Einmal**
  Das Bild wird in seiner ursprünglichen Größe einmal in der Zelle gezeigt.
- **Vertikal**
  Der Zellenhintergrund wird durch Bilder, die untereinander angeordnet sind, gefüllt.
- **Horizontal**
  Das Bild wird mehrmals nebeneinander angeordnet.
- **Kacheln**
  Der Zellhintergrund wird durch horizontale und vertikale Bildelemente gefüllt (diese Art ist der Vorgabewert).
- **Zentriert**
  Das Bild wird in der Zelle horizontal und vertikal zentriert angezeigt.
- **Größe anpassen**
  Die Anzeige des Bildes wird an die Maße der Zelle angepasst.

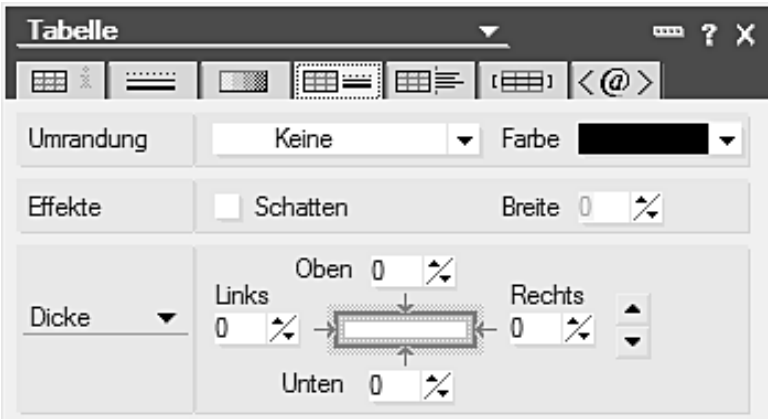
Die Verwendung von Bildern als Zellhintergrund erlaubt die Erstellung von sehr interessanten Masken. Auch reine Anzeigeseiten (z.B. für eine Web-Site) lassen sich so erstellen.



Bildanordnung in der Zelle

Über die Schaltfläche „Alle Zellen“ kann der Bildhintergrund einer Zelle für alle Zellen aktiviert werden.

Nicht nur die einzelnen Zellen, sondern auch die ganze Tabelle kann mit einem Rahmen in einer festgelegten Stärke versehen werden. Zu diesem Zweck findet sich in der Eigenschaftsbox der Tabelle ein Register für die Außenränder der Tabelle. Für die Ränder lassen sich verschiedene Effekte verwenden, die in Form von Stilen hinterlegt sind. Dazu kommt die Möglichkeit, die ganze Tabelle mit einem Schatten zu hinterlegen. Das verleiht ihr ein Aussehen, als ob sie über den anderen Gestaltungselementen schweben würde.



Tabellen-Umrandung

Im Bereich „Umrandung“ bietet der Designer viele verschiedene Stile für die Umrandung an und gestattet es, eine Farbe zu definieren.

Stile für die Tabellen-Umrandung

Die Stile bieten auch 3D-Effekte und damit mehr als man von solchen Auswahlmöglichkeiten gewohnt ist. Wird z.B. die Auswahl „Bild“ getroffen, so erscheint ein Feld, in das eine Bildressource eingetragen wird. Der Designer kachelt dieses Bild jetzt rund um die Tabelle. Es ist allerdings dabei zu beachten, dass die Stärke des Rahmens so eingestellt wird, dass das Bild auch zu sehen ist (die Stärke muss in der Regel große Werte annehmen).

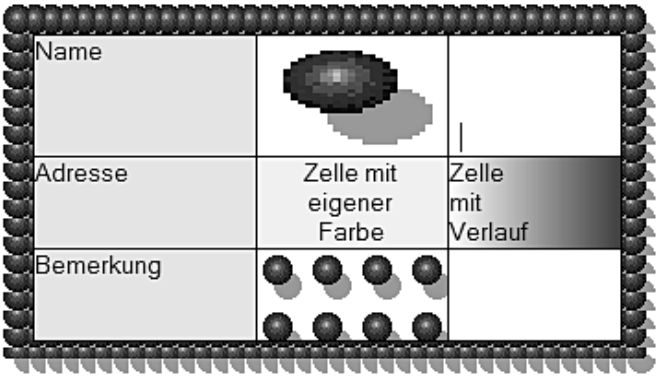


Bild als Tabellen-Umrandung

Auch die Stile „3D-Hoch2 und „3D-Tief“ liefern Effekte, die in Masken gut einsetzbar sind (z.B. für Schaltflächen).

Der Schatten einer Tabelle kann ein bzw. ausgeschaltet und seine Breite definiert werden.

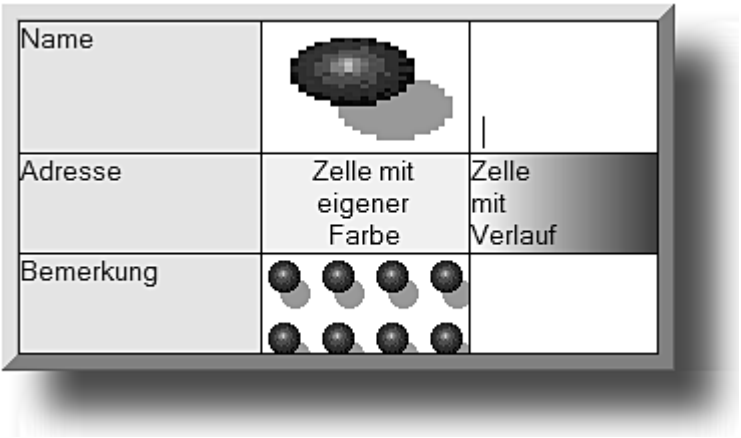
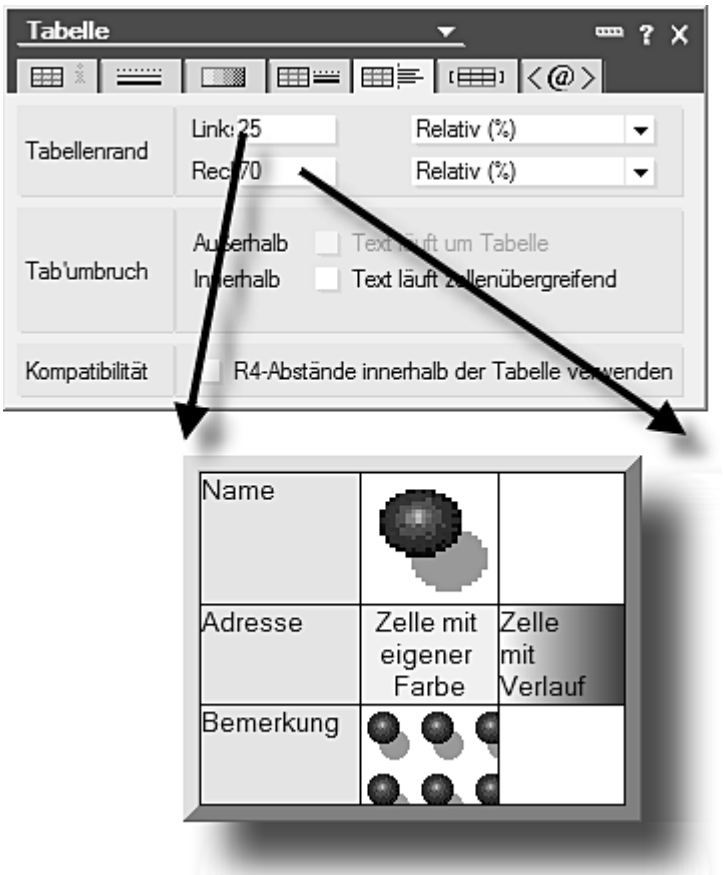


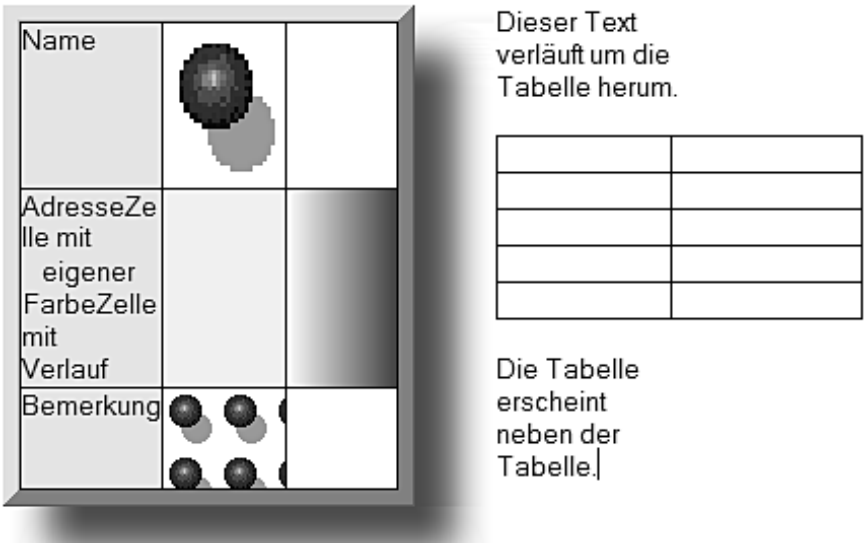
Tabelle mit 3D-Rand und Schatten

Damit die Positionierung der Tabelle nicht nur vom Seitenlayout abhängt, kann über das Register für die Tabellenränder und für den Textverlauf sowohl „absolut“ als auch „relativ“ angegeben werden; zudem wie weit sie eingerückt wird und wie breit sie sein soll. Werden absolute Werte eingesetzt, so sind sie in Zentimeter anzugeben. Die relativen Werte beziehen sich auf die Ränder.



Ränder relativ definieren

Soll neben der Tabelle Text oder ein anderes Gestaltungselement angeordnet werden, so wird im Bereich „Tabellenumbruch“ angehakt, dass der Text um die Tabelle herum laufen soll.



Textverlauf definieren

Textverlauf in Spalten

Eine mehrspaltige Anzeige wird erreicht, wenn das Feld „Text läuft zellenübergreifend“ angehakt wird. Die Funktion ist etwas gewöhnungsbedürftig und sollte nur in seltenen Fällen eingesetzt werden. Das Feld „Bei Höhe“ bestimmt, an welcher Stelle der Text in die nächste Zelle übergehen soll.

4.3.3.3.1 Eine Maske mit Tabelle erstellen

Wie bereits oben beschrieben, wird zunächst eine leere Maske angelegt, mit Standardelementen (z.B. einer Teilmaske für den Kopfbereich) gefüllt und im Hauptbereich eine Tabelle angelegt. In den Zellen der Tabelle werden dann Texte, Felder und andere Gestaltungselemente angeordnet. Über die Zeilenhöhe und Spaltenbreite werden die einzelnen Elemente ausgerichtet und über andere Tabelleneigenschaften werden Effekte hinzugefügt.

❶❷❸ Ausführungsschritte:

Beispiel einer Adressmaske, deren Felder und Feldbeschreibungen in einer Tabelle angeordnet sind:

1. Maske über die Schaltfläche oder das Menü erstellen oder eine vorhandene Maske zum einfügen der Tabelle öffnen.
2. Standardelemente einfügen (falls sie benötigt werden).
3. Tabelle mit 5 Zeilen und 2 Spalten erstellen (wie oben beschrieben). Beim Tabellentyp die Standardeinstellung „einfache Tabelle“ verwenden.
4. In der linken Spalte der Tabelle die folgenden Texte als Feldnamen einsetzen:
   Vorname
   Name
   Straße
   PLZ
   Ort

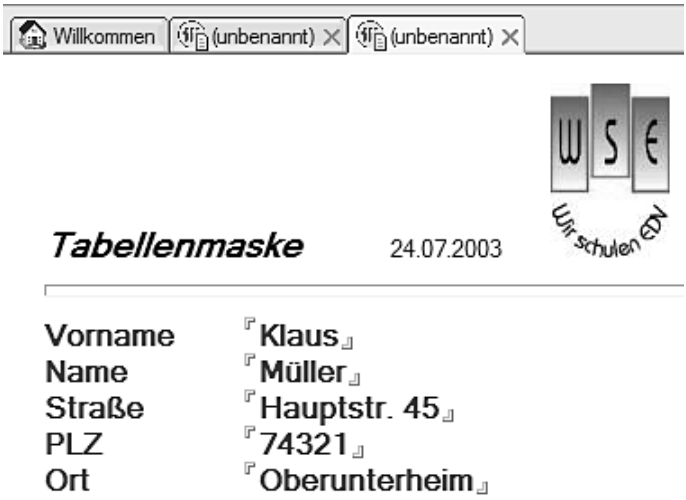
5. Auf der rechten Seite der Tabelle je ein Feld einfügen, dessen Name dem davor stehenden Text entspricht.

Vorname	Vorname
Name	Name
Straße	Strasse
PLZ	PLZ
Ort	Ort

Maske mit Tabelle und Feldern

6. Alle Zellen markieren.

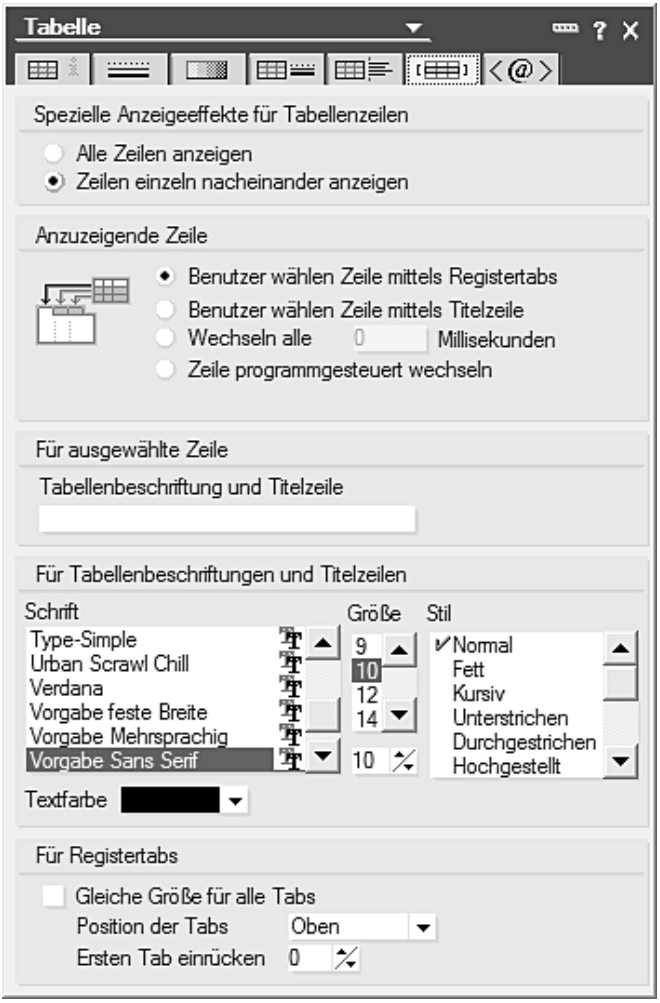
7. In den Tabelleneigenschaften das Register „Zellenumrandung“ auswählen und im Bereich „Umrandung Dicke“ die Schaltfläche „Alle auf 0“ anklicken.



Maske mit Tabelle im Notes-Client

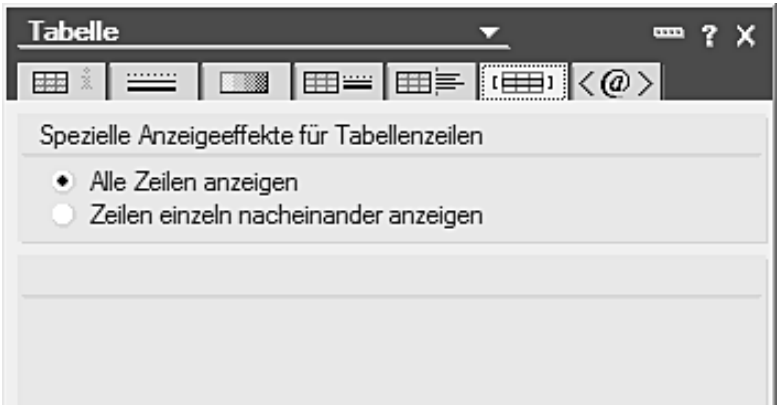
4.3.3.3.2 Tabellen mit speziellen Anzeigeeffekten

Das Register für spezielle Anzeigeeffekte ist ab Release 5 verfügbar und wurde mit Release 6 noch einmal erweitert.



Spezielle Anzeigeeffekte

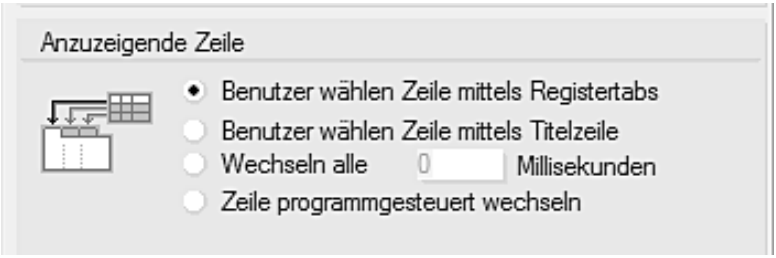
Hier finden sich Tabellenformen, die weit über das hinausgehen, was man üblicherweise unter dem Begriff „Tabelle“ vermutet. Es handelt sich um Registerdialoge, programmierbare Anzeigen und Animationen. Das grundsätzliche Merkmal der speziellen Tabellen besteht darin, dass die Zeilen einzeln angezeigt werden. Ist diese Option nicht ausgewählt, zeigt das Register nichts weiter an.



Anzeige aller Zeilen

Wird dagegen auf die Option „Zeilen einzeln nacheinander anzeigen“ umgeschaltet, so wird eine große Anzahl an weiteren Feldern sichtbar.

Der erste Bereich „Anzuzeigende Zeile“ bestimmt, wie die Anzeige der einzelnen Tabellenzeile gesteuert werden soll.

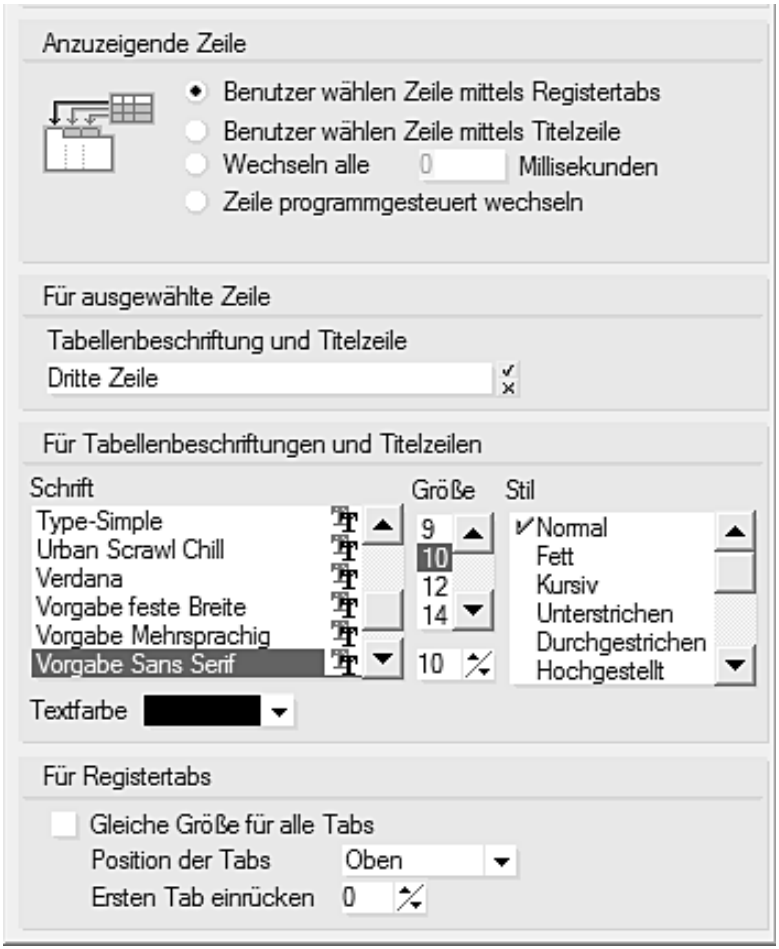


Anzeigesteuerung

Die Anzeigearten von Tabellenzeilen sind:

- Registertabelle
- Titelzeilengesteuerte Tabelle
- Zeitgesteuerte Anzeige
- Programmgesteuerte Anzeige

Für die Registertabelle werden die meisten Steuerungsfelder angezeigt. Für die Registertitel können ein Text, Schriftart und -farbe sowie Größe und Position der Registerreiter definiert werden.



Eigenschaften einer Registertabelle

Als Erstes wird der Text angegeben, der auf dem Reiter der ausgewählten Registerzeile stehen soll. Darunter können die Angaben für Schriftart, Schriftgröße und Textfarbe für den Registertitel bearbeitet werden. Im untersten Bereich können die Größe und Position der Reiter beeinflusst werden. Die Anordnung der Reiter kann dabei an jeder der vier Seiten erfolgen. Werden die Reiter des Registers nicht oben, sondern an einer der anderen Seiten angeordnet, sollten andere Effekte (wie z.B. Schatten) nicht zur Verwendung kommen.

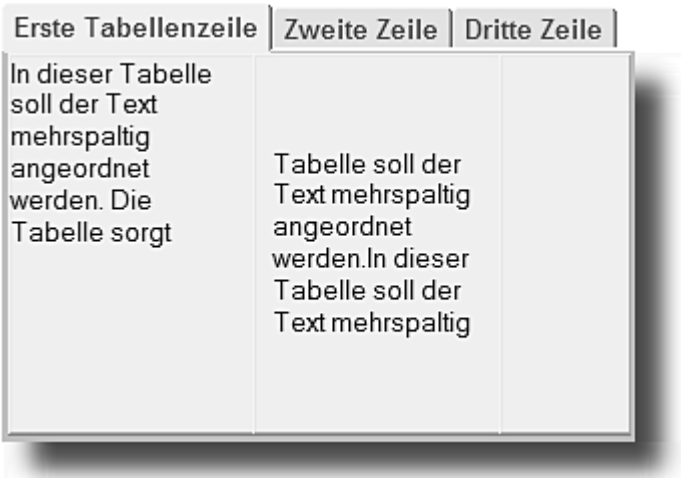


Tabelle mit Registern in der Maske

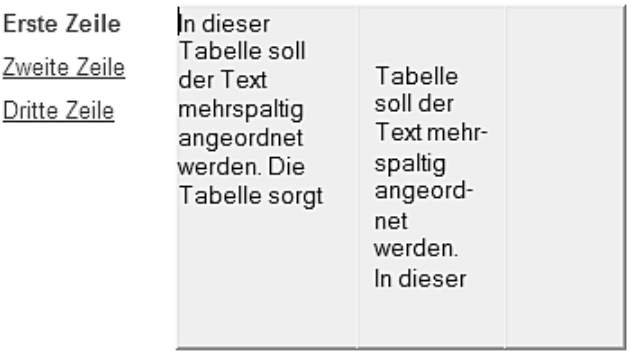
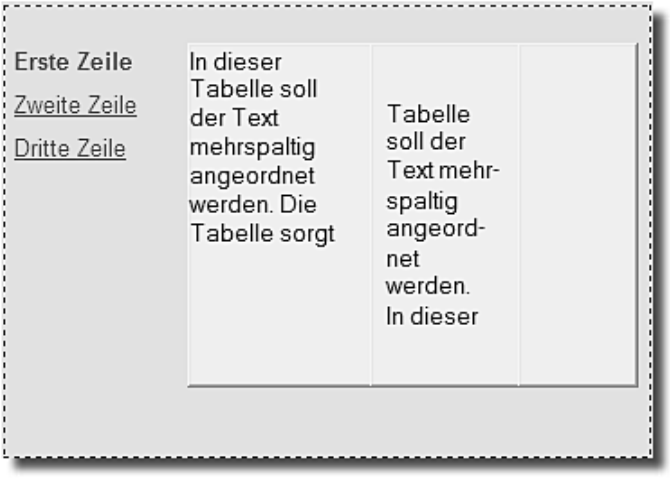


Tabelle mit Reitern auf der linken Seite

Auch die Anzeige der Reiter entspricht nicht ganz dem, was man erwarten würde. Die Titel werden einfach ohne Hintergrund und Reiterbild angezeigt. Diese Nachteile lassen sich dadurch ausgleichen, dass man die Registertabelle in eine einfache Tabelle einfügt.



Registertabelle in einer einfachen Tabelle

Um die Maske mit der einfachen Tabelle auf eine solche mit Registerreitern zu erweitern, sind folgende Schritte auszuführen:

❶❷❸ Ausführungsschritte:

Erweiterung der Adressmaske um Registerseiten, die über Reiter angesteuert werden:

1. Die gesamte Tabelle in die Zwischenablage kopieren und löschen (Ausschneiden).

2. Eine neue Tabelle erstellen mit 3 Zeilen und 1 Spalte.

3. Im Register „Spezielle Anzeigeeffekte für Tabellenzeilen“ die Option „Zeilen einzeln nacheinander anzeigen“ anklicken (die Registertabelle ist dann als Standardversion schon ausgewählt).

4. Den Cursor in das erste Register setzen.

5. Die Tabelle aus der Zwischenablage einfügen. (Die Frage „Möchten Sie die gespeicherte Tabelle in die bestehende Tabelle einfügen?“ mit „Nein“ beantworten.)

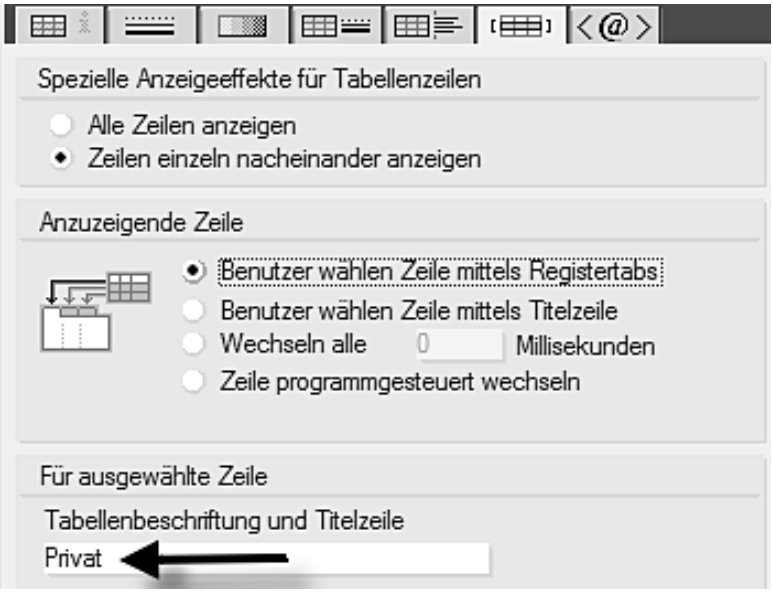
6. Den Cursor auf das zweite Register setzen.

7. Die Tabelle wieder aus der Zwischenablage einfügen. Der Designer gibt den Feldern, deren Name bereits vorhanden war automatisch neue Namen.

8. Den Cursor auf das dritte Register setzen.

9. Ein neues Feld erstellen mit dem Namen “Bemerkung” und dem Typ “Rich Text”.

10. In den Tabelleneigenschaften der Registertabelle den Reitern Titelbezeichnungen geben (jeweils den Cursor oberhalb der eingefügten Tabelle in das Register setzen). Die Namen sind: Adresse, Firma und Bemerkung.



Registertitel erstellen

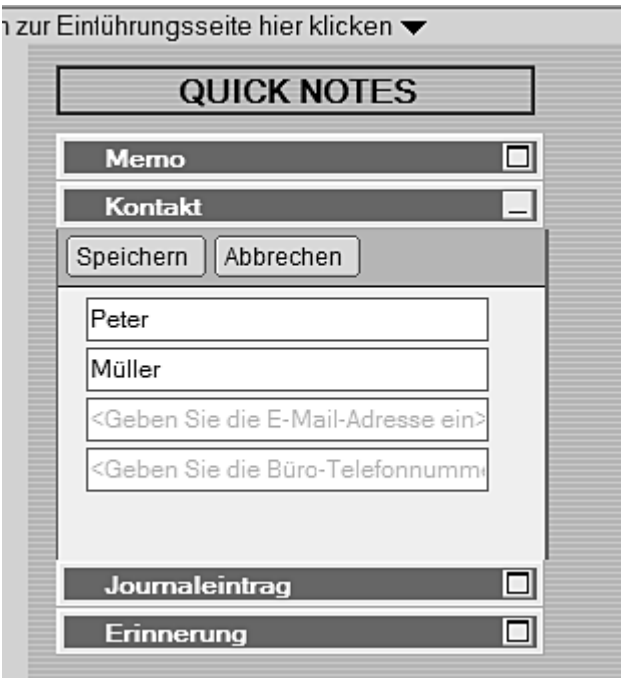
11. Um die Höhe der Registertabelle gleich zu halten, wird bei den allgemeinen Eigenschaften die Mindesthöhe auf einen Wert von ca. 6 cm gesetzt.

12. Maske im Notes-Client anzeigen lassen.

Fertige Maske mit Registertabelle

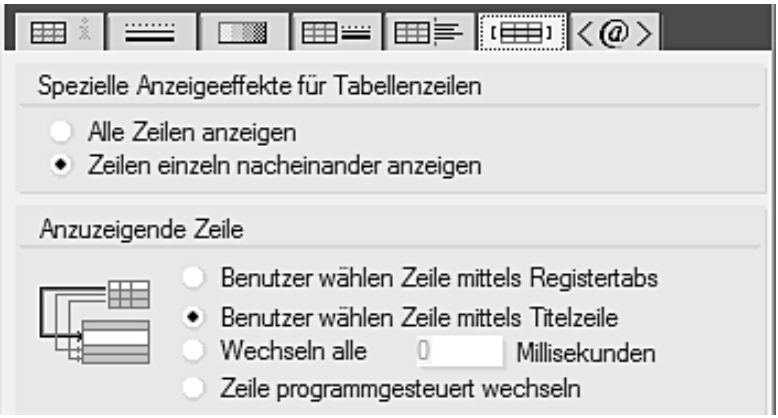
Die fertige Maske sollte im Tabellenteil aussehen – wie oben im Bild dargestellt.

Die nächste Form der Tabellendarstellung ist neu in Release 6 und trägt den Namen „Benutzer wählt Zeile mittels Titelzeile“. Es werden keine Reiter eines Registers angezeigt, sondern für jede Tabellenzeile erscheint eine Zeile, die an den Kopf eines Windows-Fensters erinnert und den Titel enthält. Die ausgewählte Tabellenzeile wird aufgeklappt und zwischen den anderen Zeilen angezeigt. Diese Tabellenform wird im Release 6 in der Willkommen-Seite für die Schnellerfassung genutzt. Dazu wird ein kleines Dreieck am linken Rand angeklickt. Dann erscheint die Tabelle „Quick Notes“ mit Memo, Kontakt, Notiz und Erinnerungstermin.



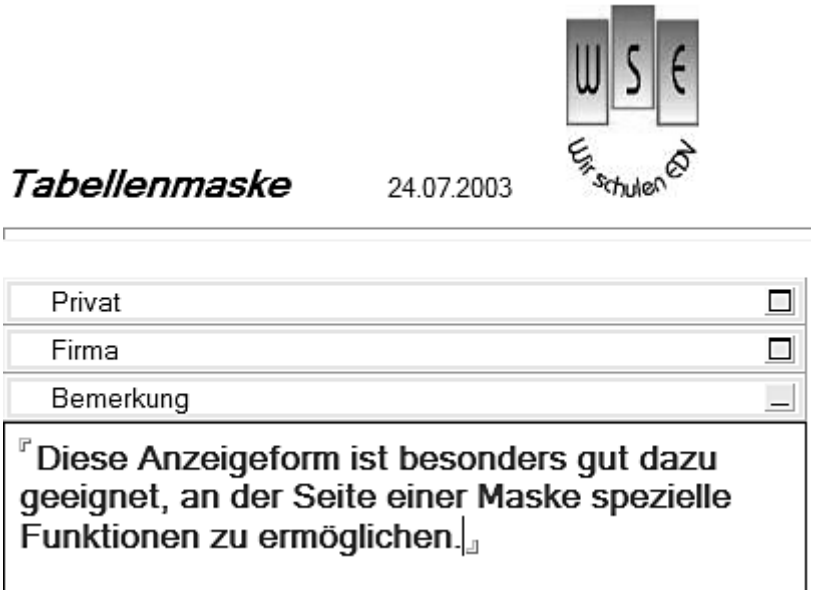
Quick Notes

Um die oben erstellte Registertabelle in diese Form zu überführen, muss lediglich in den Eigenschaften die Art der Zeilenanzeige geändert werden. Die Tabelle ändert sofort ihre Anzeigeform.



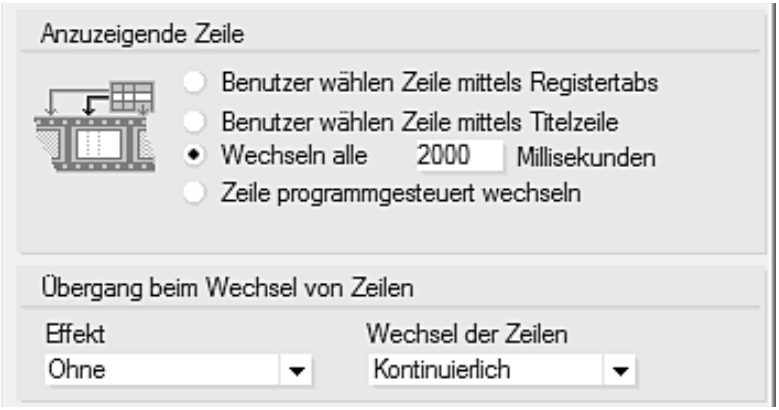
Zeilen mit Titel

Die Darstellung kann jetzt im Notes-Client kontrolliert werden.



Zeilen mit Titel im Notes-Client

Eine völlig andere Funktion hat die animierte Tabelle, bei der die Zeilen abwechselnd angezeigt werden.

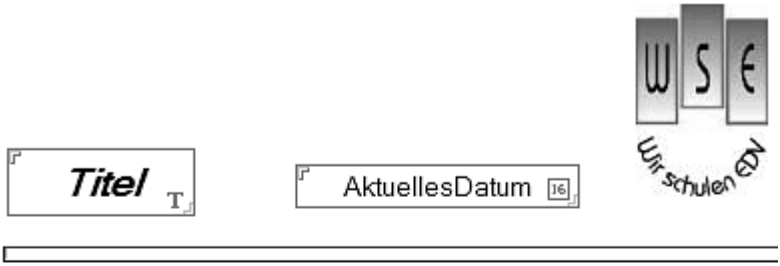


Steuerung der animierten Tabelle

Dazu wird in der ersten Zeile ein Bild oder Text gespeichert. In der zweiten Zeile wird das nächste Element eingefügt. Es werden so viele Zeilen in der Tabelle erstellt und gefüllt, wie in der Laufzeitumgebung nach einander angezeigt werden sollen. Die Anzeige der Bilder und Texte erfolgt Zeile für Zeile im eingestellten Zeitrhythmus. Es entsteht ein Effekt, wie er in der Web-Seiten-Programmierung als „Banner“ bekannt ist. Für den Übergang von einer Zeile zur anderen kann ein Effekt eingestellt werden, wie er aus Präsentationsprogrammen – wie z.B. „Microsoft Powerpoint“ – bekannt ist:

- Ohne
- Rollend
- Links nach Rechts

- Oben nach unten
- Verwischend
- Zelle für Zelle
- Explodierend
- Auflösend



Beispiel einer animierten Tabelle

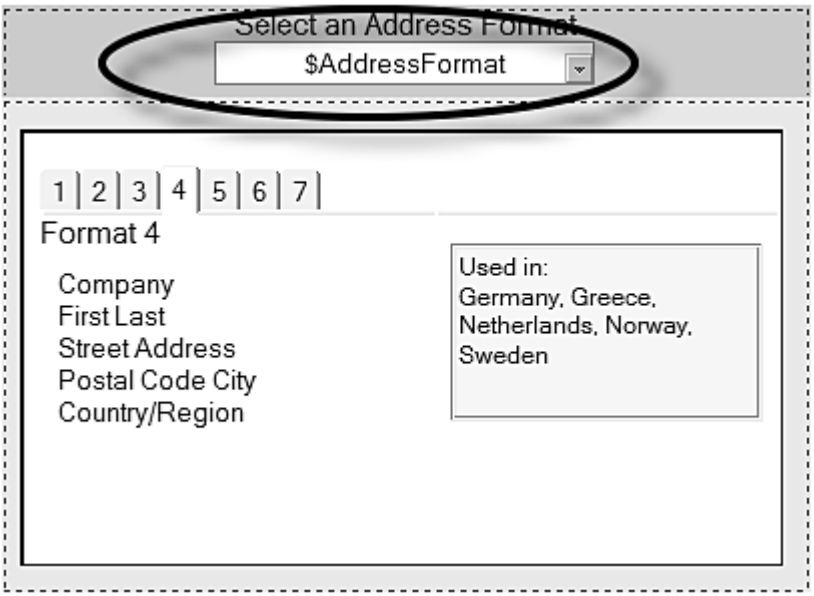
Außerdem kann das auslösende Ereignis angegeben werden. Im Normalfall erfolgt der Zeilenwechsel im angegebenen Millisekundenrhythmus. Alternativ kann der Wechsel nur einmalig oder auf Anklicken hin erfolgen.

Eine besonders interessante Version der Tabelle stellt die programmgesteuerte Tabelle dar. Hier werden die Zeilen ausschließlich oder zusätzlich zur Registersteuerung über den Inhalt eines Feldes gesteuert. Der Programmierer ändert den Inhalt des Steuerfeldes, indem er auf eine Aktion des Benutzers reagiert. Im Register für die HTML-Eigenschaften wird der Tabelle ein Name zugeordnet. Das ist gleichzeitig der Name eines Feldes, dem zusätzlich ein „$"-Zeichen voran gestellt wird.

Registertabelle mit Steuerfeld

Auch die Zeilen erhalten einen Namen. Dieser Name wird ebenfalls in die HTML-Eigenschaften eingetragen.

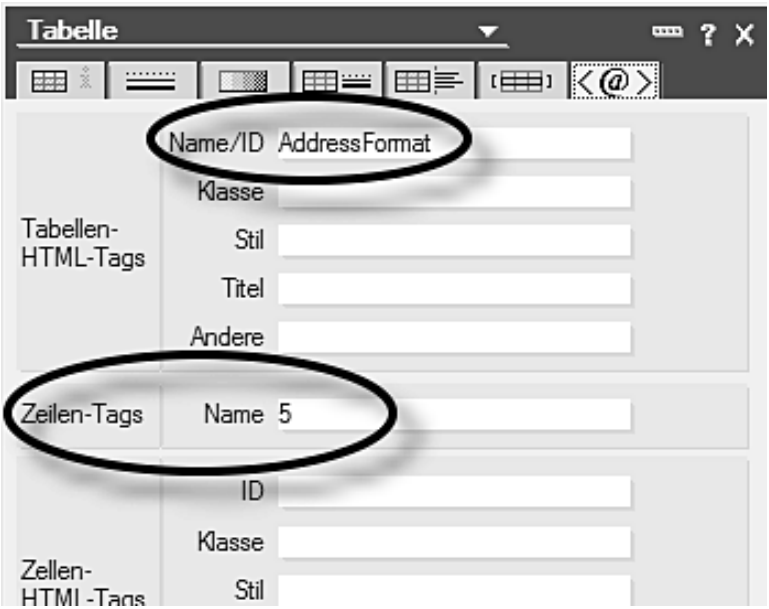
Ein schönes Beispiel für eine programmierte Tabelle findet man in der Schablone für private Adressdatenbanken in der Teilmaske „AddressFormatSubform“. Sie wird verwendet in der Maske „AddressFormatPicker“.



Registertabelle für die Wahl des Adressformats

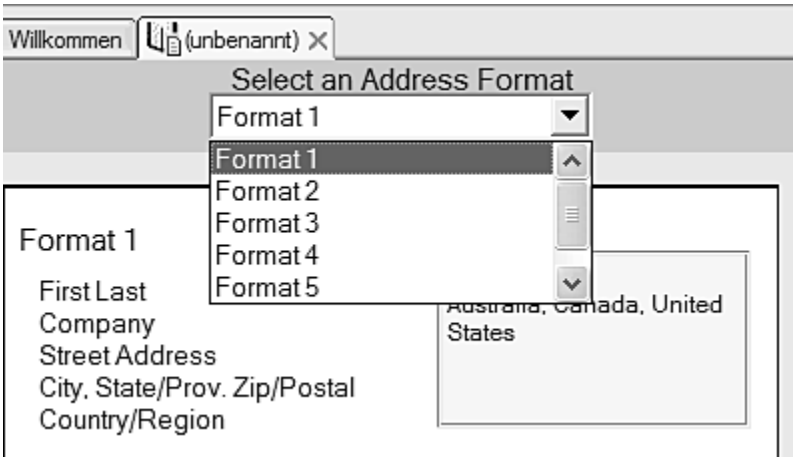
Das Steuerfeld heißt $AddressFormat und enthält eine Auswahl an Formaten. Sobald der Benutzer ein Format aus der Liste auswählt, füllt er damit das Feld mit dem entsprechenden Wert. Über die HTML-Eigenschaften der Registertabelle wird jetzt die Zeile

mit dem passenden Namen angezeigt. Die Namen sind der Einfachheit halber einfach als Ziffern von 1 bis 7 verwendet worden. Die Registerreiter werden nicht angezeigt. Der Benutzer sieht lediglich den Inhalt der Tabelle.



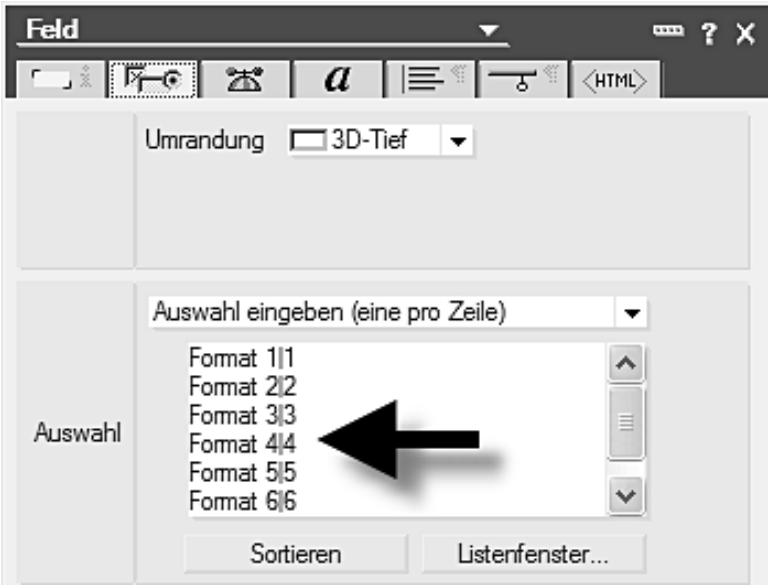
HTML-Eigenschaften

Der Wert des Feldes $AddressFormat könnte durch eine Formel oder ein LotusScript Programm-Modul verändert werden.



Anzeige der Tabelle im Notes-Client

In dieser Maske reicht die Anpassung des Wertes über die Feldliste.



Übergabewerte im Auswahlfeld

In der Definition der Auswahlwerte wurde eine Zahl als Übergabewert gewählt. Damit ist der Übergabewert gleichzeitig der Name der Zeile in der Registertabelle.

5

Ressourcen

Unter dem Punkt Ressourcen sind in der Designeranzeige Gestaltungselemente zusammengefasst, die in irgendeiner Form mehrfach verwendbar sind.

Ressourcen

Besonders interessant ist dabei die Möglichkeit, auch externe Elemente, wie Bilder und Applets, in den Prozess der Gestaltungsaktualisierung einzubinden. Folgende Ressourcengruppen können gebildet werden:

- Bilder

  Grafiken mit den Dateitypen GIF, JPEG oder BMP können importiert werden. Über das Menü „Ressourcen / Exportieren“ können die Bilder wieder ausgelagert und eventuell geändert werden. Nach der Änderung ist dann eine Aktualisierung über das Menü möglich.

- Applets

  Java-Applets werden mit ihren Dateien und Parametern unter dem Applet-Namen gespeichert.

- Teilmasken

  Masken mit Feldern und anderen Elementen, die als Bausteine für Masken verwendet werden.

- Gemeinsame Felder

  Felder, die in mehreren Masken eingesetzt werden.

- Script-Bibliotheken

  Programm-Module in LotusScript.

Unter der Position „... Andere“ verbergen sich weitere Punkte, die z.T. nur ein Element enthalten:

- Symbol

  Das Symbol der Datenbank

- Dokument „Benutzen dieser Datenbank“

  Dokument, das jede Datenbank enthält. Es kann zu einer Beschreibung der in der Datenbank enthaltenen Anwendung genutzt werden. Die Anzeige des Dokuments erfolgt über das Menü „? / Benutzen dieser Datenbank“.

- Dokument „Über diese Datenbank“

  Dieses Dokument ist ebenfalls Bestandteil jeder Datenbank. Es sollte eine Kurzbeschreibung der Datenbank enthalten.

- Datenbank-Script

  Programm, das beim Start der Datenbank ablaufen soll.

- Gemeinsame Aktionen

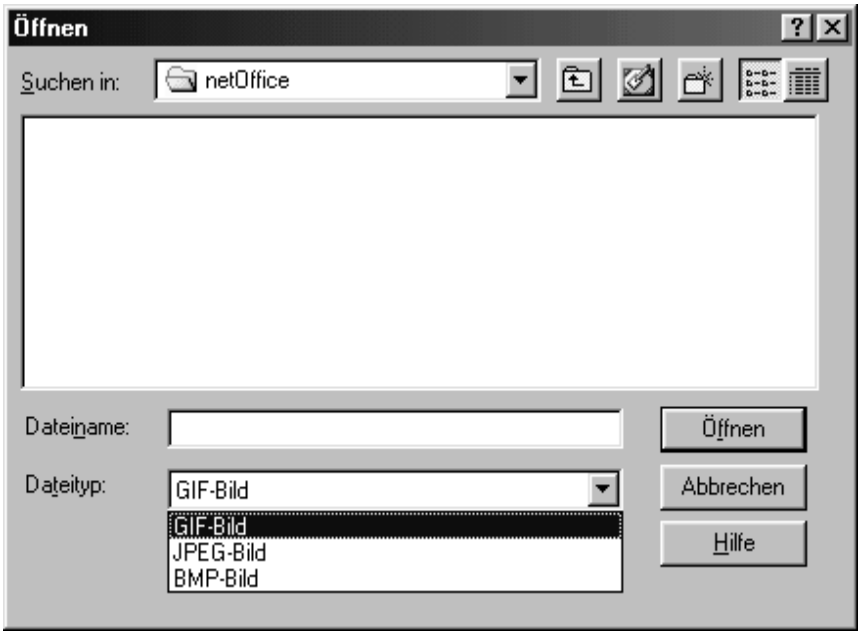
  Aktionen, die in mehreren Masken oder Ansichten eingesetzt werden sollen.

Die Ressourcen stellen nicht nur eine sinnvolle Sammlung von Gestaltungs- und Programm-Modulen dar. Durch die Methode der Gestaltungsaktualisierung und die Replikation können diese Sammlungen die Arbeit in Gruppen und Unternehmen erleichtern und vereinheitlichen.

Diese Seite ist zur Zeit nicht belegt.

5.1 Bilder verwalten mit Ressourcen

Lotus hat eine eindeutige Vorliebe für das GIF-Format bei der Bearbeitung von Ressourcen. Statt alle Bildformate anzuzeigen, werden grundsätzlich zuerst GIF-Bilder angezeigt.



Anzeige Bild-Ressourcen

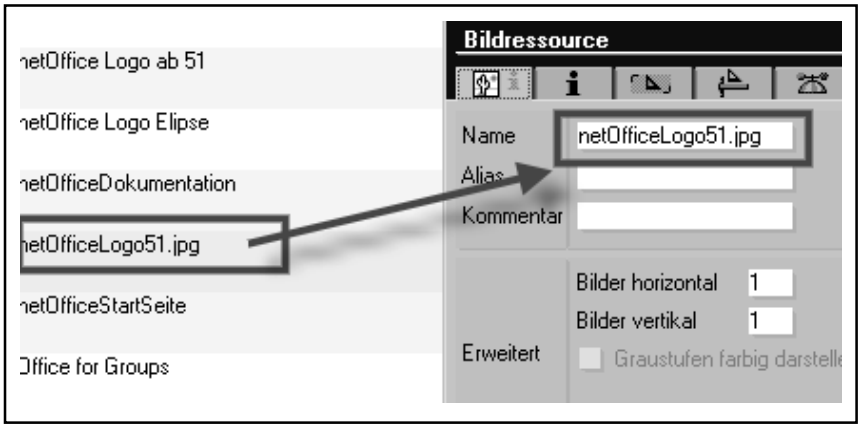
Auch bei der Verwendung anderer Formate geht es reichlich sparsam zu. Nur JPEG und BMP sind zusätzlich verfügbar. Doch wenn man von diesen Einschränkungen absieht, arbeitet es sich gut mit diesem Tool.

Diese Seite ist zur Zeit nicht belegt.

5.1.1 Bild-Ressource erstellen

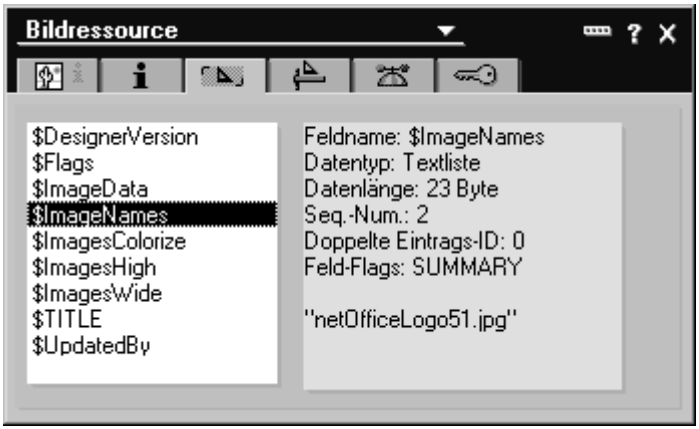
Da kein Bildverarbeitungsprogramm im Designer enthalten ist, muss ein Bild bereits vorhanden sein, wenn es als Bildressource verwendet werden soll. Über die Schaltfläche „Neue Bildressource“ erscheint der oben angezeigte Dialog und es kann ein Bild ausgewählt werden. Mit „Öffnen“ wird es in die Ressource aufgenommen und dort gespeichert. Es kann jetzt, wenn es nicht für andere Zwecke benötigt wird, als Datei gelöscht werden.

In der Liste der Bildressourcen erscheint ein neuer Eintrag, der auf Doppelklick hin ein Dialogfenster für die Ressource öffnet.



Bildressource öffnen

Im Ressourcendialog erscheint der Dateiname als Name der Ressource. Der Name kann jetzt beliebig geändert werden. Die Ressource speichert intern den Dateinamen und kann auf diese Weise jederzeit aktualisiert werden.



Dateiname der Ressource

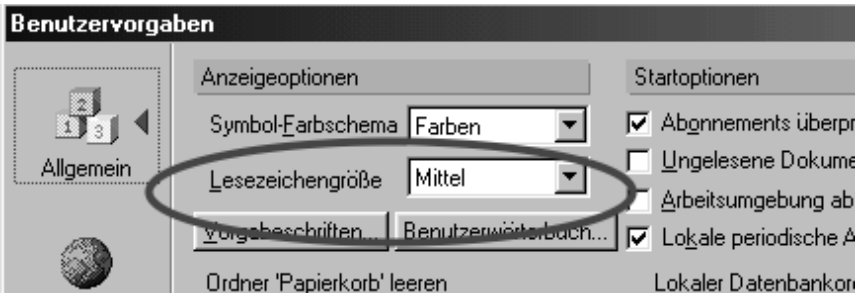
Über eine Alias-Angabe wird ein eindeutiger Name für die Verwendung in Programmen festgelegt. Der Kommentar wird in der Ressourcenliste mit angezeigt und kann zur näheren Beschreibung des Bildes dienen.

Der Bildressourcendialog

Auf der rechten Seite des Dialogs erscheint die Anzeige des gespeicherten Bildes. Neben einfachen Bildern können auch verschiedene Formate und Bilder in unterschiedlichen Zuständen verwaltet werden. Dazu dient der Abschnitt „Erweitert“.

- Unterschiedliche Größe

  In der Grafik sind drei Motive angebracht, die angezeigt werden, je nachdem, welche Größe der Benutzer in seinen Benutzervorgaben einstellt (klein, mittel oder groß).



Icongröße einstellen

Die Symbole sind mit einem Grafikprogramm so in ein Bild zu stellen, dass sie untereinander stehen und jeweils durch ein Pixel voneinander getrennt sind. Nach dem Importieren als Bildressource wird unter „Erweitert“ die Anzahl der vertikal untereinander stehenden Bilder angegeben.

Vertikale Symbole

- Variable Darstellung

  Für eine unterschiedliche Darstellung von Symbolen können Bilder horizontal nebeneinander angebracht werden. Dazu sind bis zu vier verschiedene Zustände mit eigenen Abbildungen kennzeichenbar:

  – normal
  – bei Mausberührung
  – bei Auswahl
  – bei Anklicken

Auch in diesem Fall müssen die einzelnen Bilder gleich groß und durch ein Pixel von einander getrennt sein.

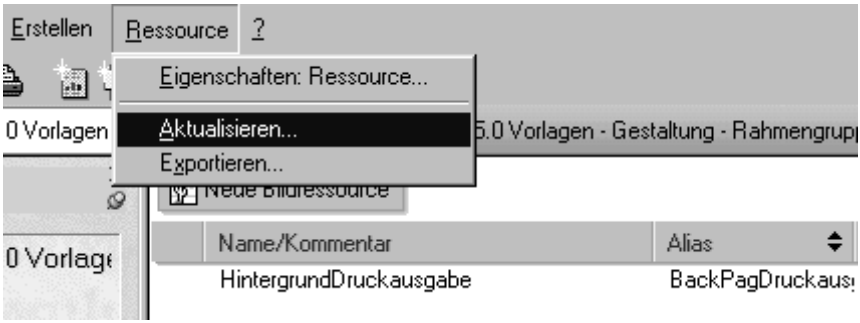
Horizontale Bildressource

Mit Hilfe der Option „Graustufen farbig darstellen“ werden Bilder nach dem Lotus-Farbschema eingefärbt verwendet.

Diese Seite ist zur Zeit nicht belegt.

5.1.2 Bild-Ressourcen aktualisieren

Soll ein Bild geändert werden, so muss zunächst festgestellt werden, ob das Bild als Grafik-Datei vorliegt. Ist das der Fall, so kann bei geöffnetem Dialogfenster im Menü „Ressourcen“ der Punkt „Aktualisieren“ ausgewählt werden.



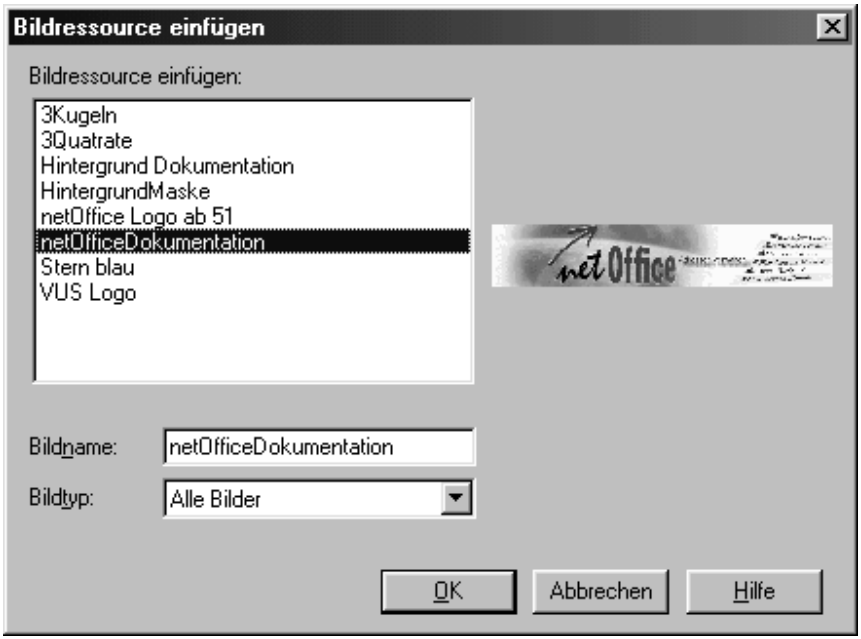
Ressource aus einer Datei aktualisieren

Der Designer zeigt einen „Datei öffnen“-Dialog, aus dem die Bilddatei zur Aktualisierung ausgewählt werden kann. Nur ein Schritt mehr ist erforderlich, um ein Bild zu aktualisieren, das nicht als Datei vorliegt. Es wird mit der Menüposition „Ressource / Exportieren“ als Datei gespeichert. Danach kann es mit einem Grafikprogramm geändert und wieder importiert bzw. aktualisiert werden. Es ist also oftmals sinnvoll, Ressourcen nicht als Grafikdateien, sondern ausschließlich in Dominodatenbanken vorzuhalten.

Diese Seite ist zur Zeit nicht belegt.

5.1.3 Bild-Ressourcen einfügen

Bilder können an den unterschiedlichsten Positionen im Rahmen des Designers in Gestaltungselemente eingefügt werden (Seiten, Seitenhintergrund, Gliederungen, Texte, Tabellen usw.). Auch der Notes-Editor bietet das Einfügen von Ressourcen an. Das setzt allerdings voraus, dass die Datenbank bereits über entsprechende Bildressourcen verfügt. Es ist nicht möglich, als Benutzer im Rahmen der Standardanwendungen Ressourcen hinzuzufügen.



Bildressource einfügen

Das ausgewählte Bild wird an der Cursorposition in das Gestaltungselement eingefügt.

Diese Seite ist zur Zeit nicht belegt.

6

Die Programmiersprachen

Lotus Domino bietet dem Anwendungsentwickler nicht nur eine, sondern ein ganzes Bündel an Programmiersprachen. Es beginnt mit einfachen Aktionen, die wie Assistenten aufgebaut sind und geht bis zu Java und JavaScript.

Folgende Programmiermöglichkeiten können eingesetzt werden:

- Einfache Aktionen
- Die Lotus-Formelsprache
- LotusScript (Basic)
- HTML
- Java
- Javascript

Jede der verschiedenen Sprachen hat eigene Schwerpunkte, Stärken und Schwächen. Da bestimmte Funktionen nur mit jeweils einer der Sprachen realisierbar sind, muss man sich mit fast jeder der aufgeführten Sprachen beschäftigen.

6.1 Einfache Aktionen

Einfache Aktionen sind im Grunde Werkzeuge, die es erlauben, mit wenigen Angaben bestimmte Arbeiten durchzuführen. Sie ähneln eher Assistenten als einer Programmiersprache und sind einer der ältesten Bestandteile von Lotus Notes. Folgende einfache Aktionen stehen zur Verfügung:

- Agent starten
- Antwort an Absender
- Aus Datenbank löschen
- Aus Ordner entfernen
- Dokument als gelesen / ungelesen markieren
- Dokumente senden
- Feld ändern
- Felder pro Maske ändern
- In Datenbank kopieren
- In Ordner kopieren
- In Ordner verschieben
- Infobrief Übersicht senden
- Mail senden
- Formel ausführen

Die meisten der einfachen Aktionen stellen Funktionen zur Verfügung, die der Benutzer ohnehin als Einzelaktion verwenden

kann. Die wichtigste Aktion, das Ändern eines Feldes, kann für einen Administrator mit entsprechenden Kenntnissen eine Möglichkeit sein, schnell Datenfehler zu berichtigen. Der Einsatz ist allerdings mit der Gefahr neuer Fehler verbunden. Insgesamt sind aus Entwicklersicht die übrigen Programmiermöglichkeiten vorzuziehen, da man sehr schnell an die Grenze der Anpassungsmöglichkeit solcher Aktionen stößt.

6.2 Die Formelsprache

Die Formelsprache als durchgängige Lotus-Makrosprache ist ebenfalls älteren Datums. Sie ist allerdings so stark in die Software integriert, dass heute noch bestimmte Aufgaben nur über die Formelsprache zu lösen sind. Die Erweiterungen des Designerwerkzeugs werden immer noch weitestgehend in die Formelsprache integriert.

Ein Formelprogramm besteht aus einem oder mehreren Befehlen und Funktionen. Diese werden nacheinander ausgeführt. Dabei kann eine Formel einmal oder mehrfach auf ausgewählte Dokumente zugreifen.

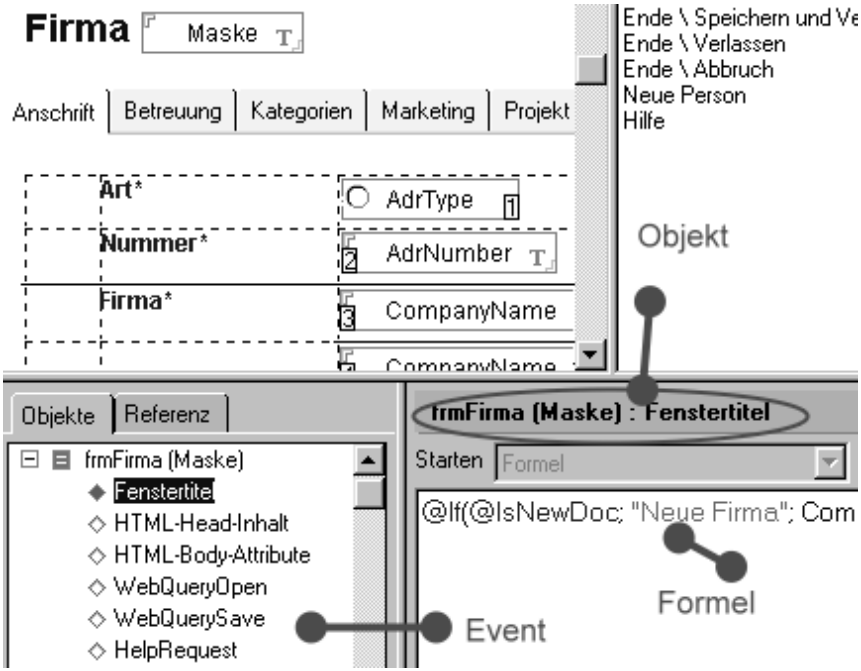
In einer Formel können nur in eingeschränktem Maße bedingte Verzweigungen eingesetzt werden. Auch für Schleifen und Sprünge stehen keine Sprachelemente zur Verfügung. Lediglich das vorzeitige Verlassen von Formeln ist möglich.

Die Formelsprache hat eine eigene Syntax mit Feldern, Konstanten, Operatoren und Standardfunktionen, die alle mit dem Zeichen @ anfangen. Dabei kann sowohl auf Datenbankfelder aus den Dokumenten als auch auf im Quellcode zusätzlich definierte Variablen zugegriffen werden. Die Syntax und die einzelnen Elemente sollen im folgenden kurz umrissen werden.

Diese Seite ist zur Zeit nicht belegt.

6.2.1 Der Formel-Editor

Formeln werden im Programmierbereich des Designer-Editors erfasst. Sie können Teil eines Events, einer Aktion oder eines Agenten sein. Das Editorfenster ist unterteilt in den Unterstützungsbereich auf der linken Seite und den Programmierbereich auf der rechten Seite. Links werden die Ereignisse (Events) zum aktuellen Objekt aufgelistet. Durch Anklicken eines Events wird rechts der dazugehörige Programmtext angezeigt.

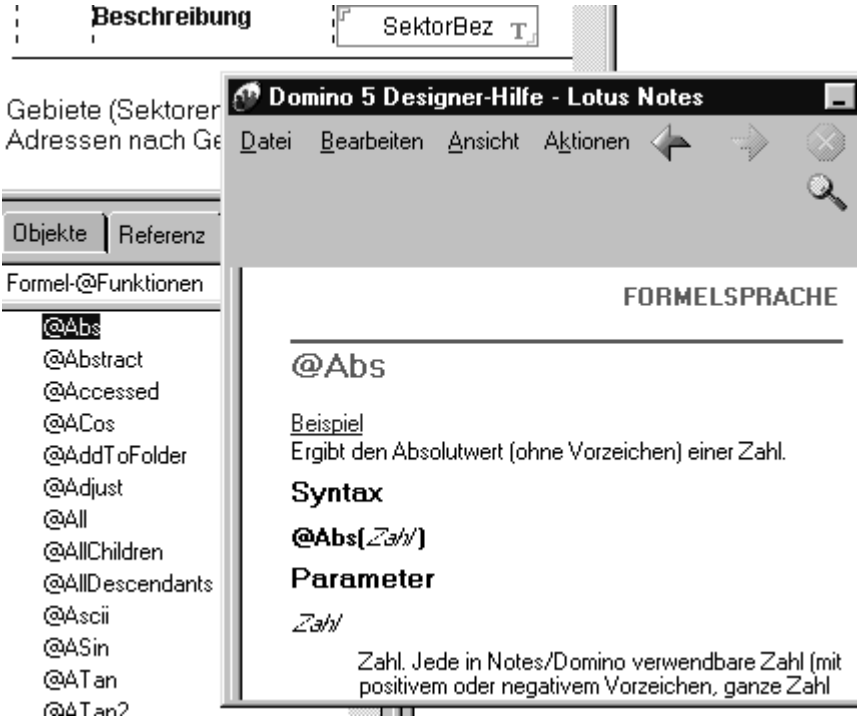


Der Formeleditor

Über das Register „Referenzen“ kann ausgewählt werden, ob andere Listen angezeigt werden sollen:

- Datenbankfelder
- @Funktionen
- @Befehle

Zu den Funktionen und Befehlen kann über die Schaltfläche „Hilfe“ eine Beschreibung angezeigt werden.



Hilfetext

Bei der Erfassung von Formeln und Befehlen im Programmierbereich unterstützt der Editor durch eine automatische Syntaxprüfung. Wenn man versucht den Editor zu verlassen oder die Schaltfläche mit dem grünen Häkchen anklickt, wird die falsche Zeile rot markiert und in der Statuszeile ein Fehler gemeldet. Der Fehler muss gefunden und behoben werden, bevor der Editor verlassen werden kann. Die am häufigsten auftretenden Fehler sind:

- falsch geschriebene Funktionsnamen der @Funktionen oder Schlüsselwörter
- fehlende oder überflüssige Klammern
- fehlendes Anführungszeichen am Ende einer Textkonstante
- kein Semikolon zwischen mehrere Befehlen

Als Empfehlung lässt sich sagen, dass man Formeln in kleinen Schritten entwerfen und so oft wie möglich durch den Syntaxchecker prüfen lassen sollte. Zu beachten ist auch, dass eine Verschachtelung der Befehle die Übersicht und die Fehlersuche behindert.

Diese Seite ist zur Zeit nicht belegt.

6.2.2 Elemente der Formeln

Wie schon der Name der Sprache es andeutet, wird die Hauptarbeit über sogenannte Formeln geleistet. Der Designer unterscheidet zwei Gruppen von Formeln

- Funktionen
- Befehle

Die Syntax für den Aufruf in allgemeiner Form lautet:

```
@ Funktionsname ( Parameter1; Parameter2; ... ; Parametern);
```

Die Befehle werden mit der Funktion @Command und einem Schlüsselwort in eckigen Klammern ([]) verwendet.

Eine Funktion besteht also aus dem Namen der Funktion gefolgt von den eventuell benötigten Parametern. Die Parameter werden von Klammern umschlossen. Wenn an eine Funktion keine Argumente zu übergeben sind, müssen die Klammern weggelassen werden. Neben den Befehlen verwenden auch viele Funktionen Schlüsselwortparameter (z.B. @Abstract, @DocMark, @GetPortList, @PickList, @MailSend, @Name und @Prompt). Eine komplette Formel sieht dann z.B. wie folgt aus:

```
@Prompt( [OK]; „Antwort”; „42”);
```

Das Semikolon kann bei der letzten Formel eines Programm-Moduls fehlen. Bestimmte Parameter in den Formeln sind optional und können weggelassen werden.

6.2.2.1 Datentypen

Die Formelsprache kennt als Datentypen Text, Zahlen, Zeit, logische Werte und Listen. Bei den Datentypen nehmen die Richtext-Felder eine Sonderstellung ein: Ihr Inhalt kann nicht einfach bearbeitet oder z.B. über die @Text( )-Funktion umgewandelt werden. Die Ausgabe von Richtext ist über eine Funktion, die als Parameter Text akzeptiert, nicht möglich.

6.2.2.2 Konstante

In der Formelsprache von Lotus Notes gibt es für die vorhandenen Typen auch entsprechende Konstanten. Texte werden in Anführungszeichen gesetzt. Zahlen werden ohne Begrenzer verwendet. Datums- und Zeitkonstanten werden mit einer eckigen Klammer angegeben. Das Format des Datums richtet sich nach der in der Systemumgebung eingestellten Sprache. Die Definition einer Konstante erfolgt durch Zuweisung eines Konstantenwertes an eine Variable:

```
FIELD Subject := „Konstanter Text”;
tmpNumPLZ := 99999;
tmpDatY2K := [13.12.2000];
```

6.2.2.3 Variablen

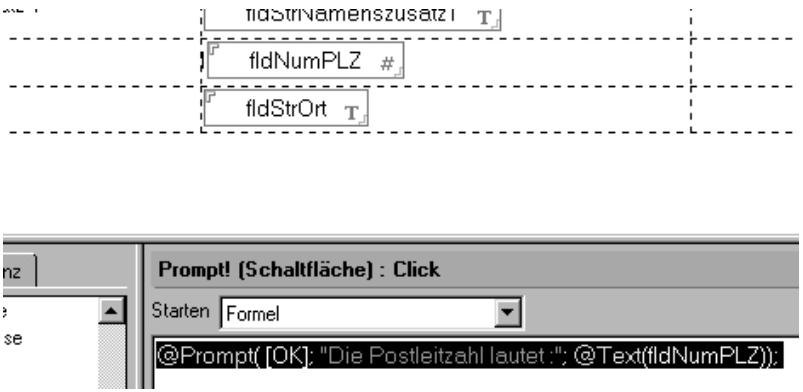
Variablen sind grundsätzlich temporär und werden ebenfalls durch Wertzuweisung definiert. Der Gültigkeitsbereich einer Variable bezieht sich nur auf das gerade aktuelle Programm-Modul. Der gleichen Variable kann kein neuer Wert mehr zugewiesen werden.

Soll der Wert einer Variable geändert werden, so muss eine neue Variable für den geänderten Wert definiert werden.

tmpWert := „Test“;
tmpWert1 := tmpWert + „ mit Wertzuweisungen“;
ergibt für tmpWert1: „Test mit Wertzuweisungen“.

6.2.2.4 Maskenfelder

Eine Formel hat Zugriff auf die Felder des aktuellen Dokuments, das gerade bearbeitet wird. Die Felder werden mit dem gleichen Namen wie in der Datenbank angesprochen. Die Funktionen erwarten als Parameter Variablen eines bestimmten Typs. Bei Bedarf müssen die Variablen konvertiert werden, bevor sie an die Funktion übergeben werden. Die Funktion ”@Prompt” benötigt z.B. einen Text als Argument. Angenommen, ein nummerisches Feld ”NumPLZ” soll als Meldung angezeigt werden, so müsste man es zunächst mit der Funktion @Text umwandeln:



Ausgabeformel mit @Prompt

Wird die Formel mit einer Schaltfläche kombiniert, so wird beim Anklicken die entsprechende Meldung am Bildschirm angezeigt.

Ergebnisbeispiel mit Schaltfläche

Um einem Feld einen Wert zuzuweisen, wird das Schlüsselwort „FIELD“ verwendet. Eine zweite Möglichkeit ist die Funktion @SetField. Während FIELD immer am Anfang einer Zeile stehen muss, kann @SetField auch innerhalb von Formeln verwendet werden. Diese Funktion setzt aber voraus, dass das Feld vorher mit FIELD initialisiert wurde:

```
FIELD Anrede := Anrede;
@if(Anrede = „“; @SetField(„Anrede“; „Damen und Herren“);
    @SetField(„Anrede“; Anrede);
```

Ein leeres Feld entspricht der Textkonstante „“ (leere Anführungszeichen).

Im folgenden Beispiel prüft die Funktion @if, ob das Feld „Betreff“ bei einem Dokument gefüllt wurde oder nicht.

```
FIELD Subject := @IF( Subject = „“; „Kein Betreff”; Subject)
```

Wenn ein Feld aus einem Dokument gelöscht werden soll, wird die Funktion @DeleteField verwendet.

```
FIELD Subject := @DeleteField
```

6.2.2.5 Operatoren

Die unter Lotus Notes verfügbaren Operatoren können genau wie in anderen Programmiersprachen verwendet werden. Dabei gibt es eine Reihe von Regeln über die Reihenfolge der Ausführung und der Prioritäten. Wichtig bei der Verwendung der Operatoren ist, dass das „=“ nur in Vergleichen verwendet wird. Bei Wertzuweisungen arbeitet die Formelsprache mit „:=“. Die Verbindung von Zeichenketten erfolgt über das „+“-Zeichen.

6.2.2.6 Schlüsselwörter

Neben FIELD gibt es noch einige weitere Schlüsselwörter. Den Schlüsselwörtern ist gemeinsam, dass sie immer am Anfang einer Zeile stehen müssen.

- DEFAULT

  Feldname := Wert

  Legt einen Standardwert für ein Feld fest. Falls in dem Dokument ein Feld <Feldname> vorkommt, wird der Wert aus dem

Dokument verwendet. Ansonsten wird das Dokument so behandelt, als gäbe es das Feld nicht und der Standardwert wird verwendet.

- ENVIRONMENT

  Variable := Wert

  Weist einer Umgebungsvariable in der Notes.ini (Windows, OS/2, UNIX) oder den Notes Preferences (Macintosh) des Benutzers einen Wert zu.

- FIELD

  Variable := Wert

  Weist einer Datenbankvariable einen Wert zu.

- REM

  „ Kommentar „

  Kommentarbefehl

- SELECT

  logischer Wert

  Mit diesem Befehl wird festgelegt, ob ein bestimmtes Dokument mit einem View oder einem Agenten bearbeitet werden darf.

Schlüsselwörter werden immer in Großbuchstaben geschrieben. Sind sie klein geschrieben, werden sie beim Speichern in große Buchstaben umgewandelt.

6.2.3 Arbeiten mit Funktionen

Die Formelsprache enthält sehr viele Funktionen und wächst durch Erweiterungen ständig. Sie beziehen sich unter anderen auf die Einsatzbereiche

- Rechnen
- Verarbeitung von Zeichenketten
- Arbeit mit Listen
- Zeit und Datum
- Anzeige von Meldungen und Daten
- Zugriff auf Systemdaten
- Datenbankzugriff

6.2.3.1 Rechnen mit Formeln

Die Formelsprache ist nicht unbedingt dazu geeignet, umfangreiche mathematische Probleme zu lösen. Für normale kaufmännische Anwendungen sind in ausreichendem Umfang Operatoren und Formeln verfügbar.

```
tmpErgebnis := tmpVar1 / 7 * (tmpVar2 + tmpVar3)
```

Spezielle Funktionen sind z.B.

- @Abs Absoluter Wert einer Zahl
- @Cos, @Sin, @Tan etc. Winkelfunktionenen

- @Exp, @Lin Logarithmusfunktionen
- @Max, @Min Größter bzw. kleinster Wert eines Zahlenvergleichs
- @Modulo Rest einer Division
- @Integer Ganzzahliger Wert einer Zahl
- @Pi 3,14...
- @Sum Summe einer Zahlenliste ermitteln

6.2.3.2 Textmanipulation

Die Formelsprache stellt mehr Funktionen zur Manipulation von Zeichenketten zur Verfügung als manche andere Programmiersprache. Hier einige Beispiele:

- @Begins

  Prüft den Anfang einer Zeichenkette auf das Vorkommen einer zweiten und liefert 1 zurück, wenn die Prüfung positiv verläuft.

  @Begins(„Domino“;“Dom“)
  Ergebnis: 1

- @Contains

  Prüft, ob eine Zeichenkette in einer anderen enthalten ist.

  @Contains(„Domino“;“ino“)
  Ergebnis: 1

- @Left

  Liefert den linken Teil einer Zeichenkette mit der angegebenen Anzahl von Zeichen.

  @Left(„Domino“;2)
  Ergebnis: Do

- @LowerCase

  Wandelt Groß- in Kleinbuchstaben um.

- @Char

  Übersetzt die Codepage-Nummer in das entsprechende Zeichen.

  @Char(66)
  Ergebnis: B

- @Length

  Liefert die Länge einer Zeichenkette zurück.

- @Repeat

  Liefert eine Zeichenwiederholung in einer Zeichenkette.

  @Repeat(„Domino“; 2)
  Ergebnis: DominoDomino

- @NewLine

  Fügt einen Zeilenumbruch ein.

  „Lotus“ + @NewLine + „Domino“
  Ergebnis: Lotus Domino

Auch für die Konvertierung von Datentypen enthält die Formelsprache Funktionen:

- @IsText

  Prüft, ob ein Wert den Datentyp Text hat.

- @Text

  Wandelt Zahlen in Text um.

  @Text(324,12)
  Ergebnis: „324,12“

- @TextToNumber

  Wandelt Text in Zahlen um.

- @TextToTime

  Wandelt Text in einen Zeit-/Datumswert um.

  @TextToTime(„08.01.2000 17:23“)
  Ergebnis: 8.1.2000 17:23:00

Die Funktionen können auch ineinander verschachtelt werden (z.B. @UpperCase(@Left(„Domino“; 3) + „platz“) ergibt DOMPLATZ). Diese Methode vermeidet das Einfügen neuer Variablen. Welche Methode eingesetzt wird, sollte sich an der Lesbarkeit des Programmtextes orientieren.

6.2.3.3 Listenverarbeitung

Der Typ Listen wird für eine Anzahl gleicher Elemente verwendet. In anderen Sprachen werden Listen auch als Array bezeichnet. Die Formelsprache stellt für die Bearbeitung von Listen eine Reihe von speziellen Funktionen zur Verfügung. Um zum Beispiel die Anzahl der Elemente in einer Liste zu ermitteln, kann die Funktion @Elements verwendet werden.

- @Word

  Liefert den Wert des n-ten Wortes zurück, das durch das angegebene Begrenzerzeichen getrennt ist.

  @Word(„Lotus Domino und Notes“; „ „; 3)
  Ergebnis: und

- @Member

  Hat die Position des angegebenen Wortes als Ergebnis.

  @Member(„Zwei“; „Eins“ : „Zwei“ : „Drei“)
  Ergebnis: 2

- @Subset

  Liefert einen Listenausschnitt.

  @Subset(@DbName; 1)
  Ergebnis: Name des aktuellen Servers
  @Subset(@DbName; -1)
  Ergebnis: Name der aktuellen Datenbank.

  Die negative Zahl bewirkt, dass @Subset von rechts nach links arbeitet.

- @Implode
  Wandelt eine Liste in eine Zeichenkette um.

  @Implode(„Eins“:“Zwei“:“Drei“; „-“)
  Ergebnis:

Test mit Implode

6.2.3.4 Datum und Zeit

Für die Bearbeitung von Datumswerten enthält die Formelsprache Funktionen, die spezielle Datumswerte liefern oder zur Ermittlung neuer Datumswerte genutzt werden können. Dazu kommen Funktionen, die Daten des Systems liefern.

- @Year

  Liefert das Jahr aus einem Datumswert.

  @Year(@Today))
  Ergebnis: 2000 (als Zahl)

- @Yesterday

  Ergibt das gestrige Datum

- @Adjust

  Berechnet ein Datum aus einem anderen Datum.

  @Adjust(@Today; 0; 0; 14; 0: 0; 0)
  Ergebnis: Datum des Tages, der 14 Tage später als heute liegt.

- @Created

  Erstellungsdatum eines Dokuments

6.2.3.5 Programmsteuerung

Die Steuerung von Abläufen in einem Programm-Modul ist in der Formelsprache nur in geringem Umfang möglich. Die Befehle erlauben keine Schleifen oder Sprünge, sondern im günstigsten Fall ein vorzeitiges Beenden des Programm-Moduls.

- @if

  Ausführung von Befehlen in Abhängigkeit von einer Bedingung in der Form

```
@if(Bedingung; Befehl für Bedingung erfüllt;
    Befehl für Bedingung nicht erfüllt)
@if(tmpVar=1;
   @Prompt([ok]; „Anzeige“; „Eins“);
   @Prompt([ok]; „Anzeige“; „Nicht Eins“)
```

- @Do

  Möglichkeit, mehrere Befehle in einer @if-Anweisung zu verarbeiten.

```
tmpString := @Implode(„Eins“:“Zwei“:“Drei“;“-“);
tmpStringZwei := „Zwei“;
@if(tmpString != „Zwei“;
    @Do(@Set („tmpStringZwei“; „Drei“);
    @Prompt([ok]; „Programmsteuerung“;
            tmpStringZwei));
    @Prompt([ok]; „Fehler“; tmpStringZwei));
```

  Ergebnis:

Verwendung von @Do zur Programmsteuerung

- @Return

  Bricht die Ausführung einer Formel ab und beendet das Programm-Modul ohne Ausführung der restlichen Formeln.

```
@if(tmpVar = „Zwei“; @Return(„Zwei“);
    @Prompt([ok]; „Anzeige“; „Es geht weiter...“)
```

- @Failure

  Wird bei Eingabeprüfungen verwendet. Die Ausführung wird unterbrochen und eine Fehlermeldung ausgegeben.

  @if(Land = „“; @Failure(„Das Feld Land muss gefüllt sein!“);
  @Success)

6.2.3.6 Dokumente bearbeiten

Bei der Bearbeitung von Dokumenten oder Feldern ist es häufig notwendig, auf Dokumenteneigenschaften zurückzugreifen oder eine Aktion mit dem aktuellen Dokument durchzuführen. Beispiele für solche Funktionen sind:

- @AddToFolder

  Fügt das aktuelle Dokument einem Ordner hinzu und entfernt es aus einem anderen Ordner, wenn dieser angegeben wird.

  @AddToFolder(„Kunden“; „ „)

- @DeleteDocument

  Löscht in Agenten das aktuelle Dokument.

- @IsNewDoc

  Gibt 1 zurück, wenn das Dokument neu ist.

  @if(@IsNewDoc; „Neues Dokument“; Name)

- @IsDocBeingEdited

  Prüft den Editiermodus.

  @if(@IsDocBeingEdited; @Date; @Return(„“)

6.2.3.7 Mit Umgebungsvariablen arbeiten

Die Formelsprache verfügt über ein Schlüsselwort und Funktionen, um Werte in der Datei notes.ini zu verwalten. Diese Möglichkeit, Werte nicht temporär, sondern dauerhaft zu hinterlegen, birgt allerdings eine Schwierigkeit. Der Notes-Client speichert die Inhalte dieser Datei für eine Sitzung. Erst wenn das Programm beendet und Notes wieder gestartet wird, stehen neue Werte zum Abruf bereit.

- ENVIRONMENT

  Setzt einen Wert in der notes.ini fest.

  ENVIRONMENT „Benutzer“ := @Name([CN]; @UserName)
  Ergebnis: Benutzer=Peter Muster

- @SetEnvironment

  Hat die gleiche Wirkung wie ENVIRONMENT.

- @Environment

  Übernimmt einen Wert aus der notes.ini und ändert den Wert bei Bedarf ab.

  @Environment(„Benutzer“; @Name([CN]; @UserName)

6.2.3.8 Daten übernehmen und bearbeiten

Wenn in einem Programm-Modul auf Daten aus anderen Dokumenten, aus anderen Datenbanken oder gar auf externe Daten zugegriffen werden soll, so kann das, wenn auch eingeschränkt, über Formelfunktionen erfolgen.

- @DbColumn

  Ermittelt die Werte einer Spalte in einer Ansicht oder einer ODBC-Datenquelle und gibt eine Liste aller Spaltenwerte der Dokumente bzw. Datensätze zurück.

- @DbLookUp

  Ermittelt anhand eines Schlüsselwertes ein Dokument bzw. einen Datensatz aus einer Ansicht oder einer ODBC-Datenquelle.

- @DbName

  Liefert den aktuellen Server und die aktuelle Datenbank als Liste.

- @DocFields

  Gibt eine Liste der Felder eines Dokuments zurück.

6.2.3.9 Sonstige Funktionen

Einige weitere Funktionen sind bei der Bearbeitung von Notes-Daten wichtig und hilfreich. So können Meldungen z.B. mit der Funktion @Prompt angezeigt werden. Diese Funktion bietet aber noch viele andere Verwendungsmöglichkeiten (Auswahllisten, Dateien zur Auswahl oder eine Kennwortabfrage). Weitere Dialogfunktionen sind:

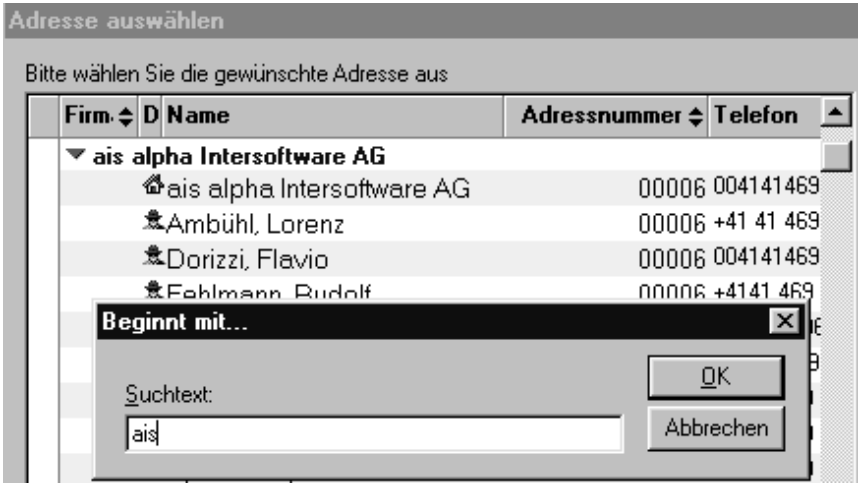
- @Dialogbox

  Zeigt ein Dialogfenster am Bildschirm an, in dem Felder des aktuellen Dokuments mit Hilfe einer anderen Maske angezeigt werden können. Diese Funktion wird häufig dazu genutzt, nur einen Teil der Maskenfelder zur Bearbeitung anzuzeigen.

Dialogbox im persönlichen Journal

- @Picklist

  Zeigt eine Liste von Dokumenten in einem Auswahlfenster an. Es kann sich dabei um beliebige Dokumente, Personen, Räume, Ressourcen oder Ordner handeln. Wird eine Zeichenfolge eingegeben, so wird der Fokus auf die erste Zeile gesetzt, die die Zeichenfolge enthält. Das Ergebnis der Funktion ist ein Wert des ausgewählten Dokuments.



Die Funktion @Picklist

Diese Seite ist zur Zeit nicht belegt.

6.2.4 Befehle

Befehle stellen Aktionen bereit, die über die Funktion @Command und unter Angabe eines Schlüsselwortes aufgerufen werden. Viele der Befehle entsprechen Positionen in den verschiedenen Menüs (z.B. FileSave zum Speichern).

- [Compose]

  Öffnet ein neues Dokument mit der angegebenen Maske und schaltet den Bearbeitungsmodus ein.

  ```
  @Command([Compose]; „frmAdressen“)
  ```

- [Execute]

  Startet ein externes Programm

  ```
  @Command([Execute]; „c:\\Programme\\Word.exe“; „c:\\Eigene Dateien\\Text.doc“).
  ```

  Die Backslash-Zeichen (\) müssen doppelt geschrieben werden!

- [FileCloseWindow]

  Schließt das aktuelle Fenster.

  ```
  FIELD SaveOptions := 0;
  @Command([FileCloseWindow])
  ```

  Schließt das Fenster ohne Abfrage, ob gespeichert werden soll.

- [FileSave]

  Speichert das aktuelle Dokument.

- [ToolsRunMacro]

  Startet einen Agenten.

Diese Seite ist zur Zeit nicht belegt.

6.2.5 Zusammenfassung

Die Formelsprache beinhaltet eine umfangreiche Sammlung von Funktionen und Befehlen, mit denen einfache Programm-Module erstellt werden können. Je umfangreicher die Programm-Module werden, desto schwieriger wird es, den Überblick zu behalten. Durch das Fehlen von Schleifen- und Sprungbefehlen ist es oft nicht möglich, einer Aufgabenstellung gerecht zu werden.

Da bestimmte Objekte und Ereignisse nur mit Hilfe der Formelsprache programmierbar sind, kommt man um ihre Verwendung nicht herum.

Diese Seite ist zur Zeit nicht belegt.

6.2.6 Erweiterungen der Formelsprache in Lotus Domino und Notes 6

Wer die Entwicklungsumgebung von Lotes Notes in den letzten Jahren miterlebt hat, hat sich immer wieder gefragt: Wird die Formelsprache endgültig abgelöst oder nicht? Und wenn die Formelsprache abgelöst wird, welche der Sprachen, die inzwischen von Domino und Notes unterstützt werden, kann die Nachfolge antreten?

Die Hinwendung zu den Internetstandards hat ihre Spuren hinterlassen. Viele Elemente der Entwicklungsumgebung tragen dem Rechnung. Die Formelsprache wurde – im Gegensatz zu LotusScript – in die Welt des Internets mitgenommen und über den HTTP-Task des Domino-Servers in den Browser übertragen. Damit wurde die Formelsprache aufgewertet und erneut in den Mittelpunkt der Notes-Entwicklung gestellt.

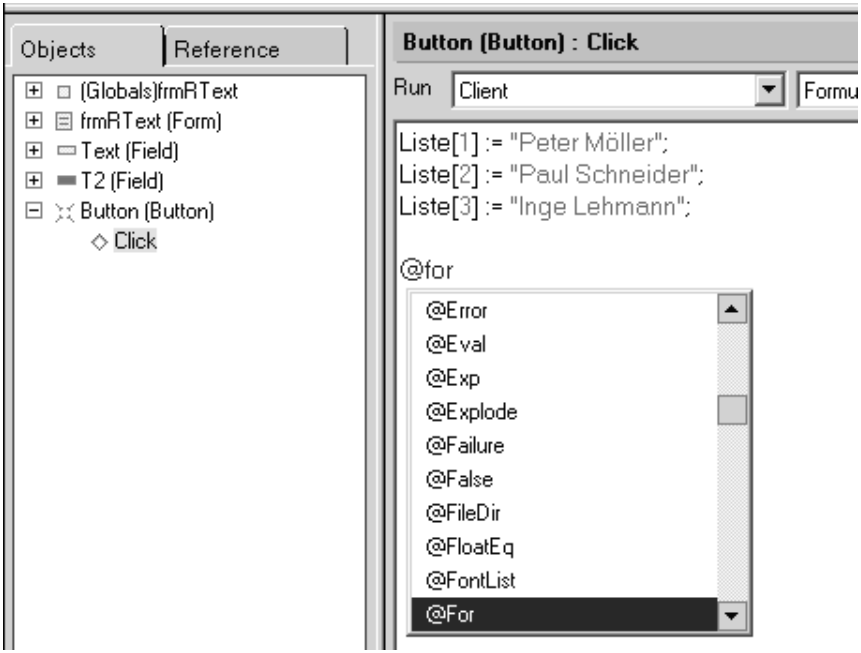
Wie auch immer die Entwicklung bei IBM Lotus weitergeht, wer damit gerechnet hat, das nächste Release bringe keine wesentliche Weiterentwicklung im Bereich der Formelsprache, der hat sich gewaltig getäuscht. Die Formelsprache bleibt ein wichtiges Tool im Rahmen der Anwendungsentwicklung unter Lotus Domino und Notes. Es wurden bestehende Formeln erweitert und wichtige neue Formeln in die Sprache eingebracht. Die wichtigsten Neuerungen sind:

- **Schleifenverarbeitung** wird mit den Funktionen „@For", „@While" und „@DoWhile" eingeführt.
- Die **Wiederbenutzung temporärer Variablen** (das war bisher nur über die Funktion „@Set" möglich) ist zulässig.

Mai 2002 / 9. EL

- **REM-Anweisungen** können geschweifte Klammern verwenden.
- **Listen** können über einen Index angesprochen und mit neuen Funktionen bearbeitet werden.
- **Zuweisungen** können ineinander genestet sein.
- **Felder** können in Funktionen verwendet werden, ohne den genauen Feldnamen zu benutzen.
- Die **Fehlerbehandlung** wird mit Hilfe von „@ifError" verbessert.

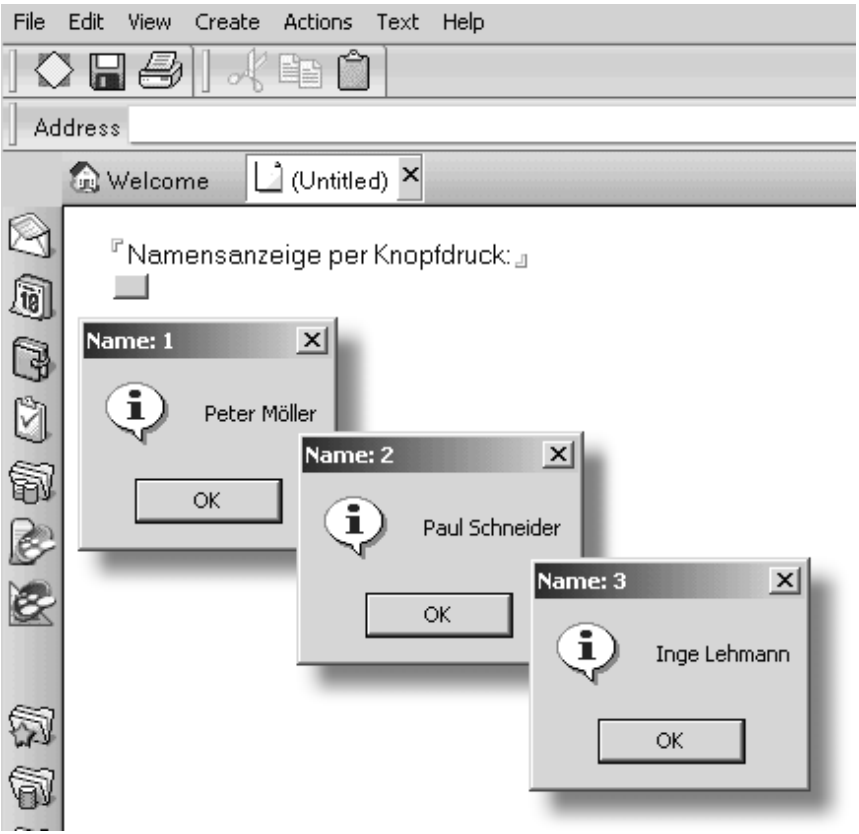
Insbesondere die Schleifenverarbeitung und die Wiederverwendung von temporären Variablen erweitern den Einsatzbereich der Formelsprache erheblich. Die integrierte Entwicklungsumgebung wurde auch im Formelbereich so erweitert, dass Funktionen und Befehle, die der Benutzer einträgt, automatisch ergänzt werden können.



Automatische Ergänzung von Formeln

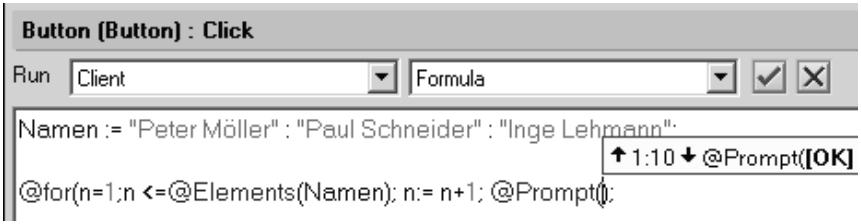
Je nach der eingegebenen Anzahl von Buchstaben wird die passende Auswahl an Schlüsselwörtern angezeigt und die Auswahl mit Hilfe der Enter-Taste oder einen Mausklick ermöglicht. Das ausgewählte Schlüsselwort wird dann an der Cursorposition eingefügt.

Für die Verwendung der Funktion oder des Befehls wird als Nächstes die Syntax über einen Hilfetext eingeblendet.



Eingeblendete Syntax

Sind mehrere Versionen der Funktion oder des Befehls möglich, kann über die Pfeiltasten durch den Hilfetext geblättert werden.



Hilfetext mit Funktionsvarianten

Damit wird das lästige Nachsehen in den Hilfetexten häufig überflüssig und die Arbeit des Entwicklers wesentlich beschleunigt.

6.2.6.1 Schleifenverarbeitung

Die Syntax der @For-Schleife erinnert an Java und C. Es wird zunächst der Startwert des Schleifenzählers und dann die Laufbedingung angegeben. Die dritte Angabe bezieht sich auf den Erhöhungswert. Danach folgen die Anweisungen, die während eines Schleifendurchlaufes ausgeführt werden sollen. Die genaue Syntax lautet:

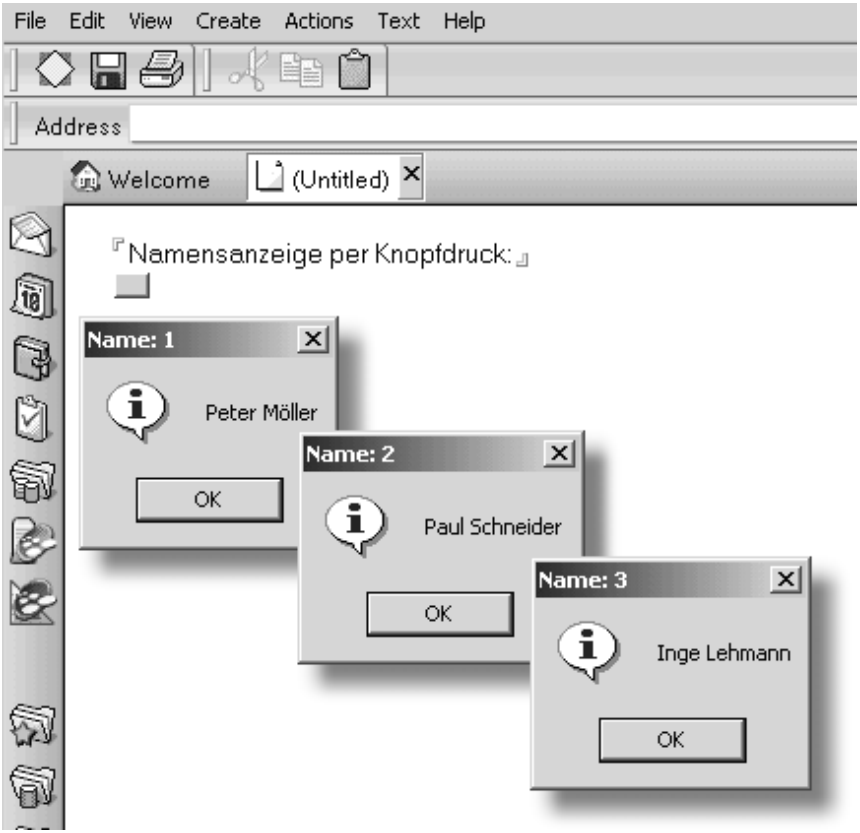
@For(Initialisierungswert; Laufbedingung; Erhöhungswert; Anweisungen)

Der Initialisierungswert wird durch eine Anweisung mit einer Zahlzuweisung zur Zählvariablen definiert. Die Laufbedingung besagt, bei welchen Werten der Zählvariablen die Schleifen-

verarbeitung durchgeführt werden soll. Die Erhöhung der Zählvariablen wird ebenfalls durch eine Anweisung erreicht. Folgendes Beispiel zeigt die Funktionsweise der @For-Schleife:

Programmtext

```
REM {Kommentare können in geschweifte Klammern gestellt werden.
    Das erleichtert das Ausblenden von Anweisungen als Kommentar, wenn die Anweisungen bereits Anführungszeichen als Bestandteil enthalten.

Zunächst werden die Elemente einer Liste mit Werten gefüllt.};

Namen := „Peter Möller” : „Paul Schneider” : „Inge Lehmann”;

REM {In der @For-Schleife werden die Namen über die @Prompt-Funktion angezeigt.};

@For(i := 1; i <= @Elements(Namen); i:= i + 1;
@Prompt([OK]; „Name: „ + @Text(i); Namen[i]))

REM {Das Ergebnis hat folgendes Aussehen: };
```



Namensanzeige

In dem kleinen Beispielprogramm wurde neben der @For-Schleife auch die direkte Verwendung von Listenelementen über ihren Index (Namen[i]) gezeigt.

Die @DoWhile-Schleife hat ihre Bedingung am Ende des Schleifendurchlaufs. Es werden also alle Anweisungen innerhalb der Schleife mindestens einmal ausgeführt. Die Syntax der @DoWhile-Schleife ist:

@DoWhile(Anweisungen; Abbruchbedingung)

Das oben stehende Programmbeispiel lässt sich mit Hilfe der @DoWhile-Funktion wie folgt realisieren:

Programmtext

```
Namen := „Peter Möller” : „Paul Schneider” : „Inge Lehmann”;

REM {Zunächst wird geprüft, ob ein Schleifendurchlauf erwünscht
     ist. Enthält die Liste keine Namen, so wird die Ausführung des
     Programms beendet.};

@if(@Elements(Namen) = 0; @Return(0); ,,,,);

REM {In der @DoWhile-Schleife werden die Namen über die
     @Prompt-Funktion angezeigt. Die Zählvariable wird von der
     Schleifenfunktion nicht automatisch verwendet. Neu ist, dass
     einer temporären Variablen mehrmals ein Wert zugewiesen wer-
     den kann, ohne spezielle Funktionen zu verwenden. }

i := 1;

@DoWhile(@Prompt([OK]; „Name: „ + @Text(i); Namen[i]);
     i = i + 1;
     i <= @Elements(Namen))

REM {Das Ergebnis ist mit dem oben gezeigten identisch.};
```

Umgekehrt steht bei der @While-Schleife die Abbruchbedingung am Anfang – also vor den Schleifenanweisungen. Damit werden diese Anweisungen, wenn die Bedingung am Anfang nicht erfüllt ist, nie ausgeführt.

Die Syntax der @While-Schleife lautet entsprechend:

@While(Laufbedingung; Anweisungen)

Auch diese Funktion kann für die Namensanzeige unseres Beispiels verwendet werden:

Programmtext

```
REM {Da die Laufbedingung am Anfang der @While-Schleife steht,
    ist keine Prüfung darauf erforderlich, ob die Liste überhaupt Werte
    enthält.};

Namen := „Peter Möller” : „Paul Schneider” : „Inge Lehmann”;
i := 1;

@While(i <= @Elements(Namen);
    @Prompt([OK]; „Name: „ + @Text(i); Namen[i]);
    i = i + 1);

REM {Das Ergebnis ist wieder mit dem ersten identisch.};
```

Eine wichtige Funktion zur Durchführung der Schleifenoperationen ist die Funktion @Elements, die jeweils die Anzahl der Listenelemente zurückgibt.

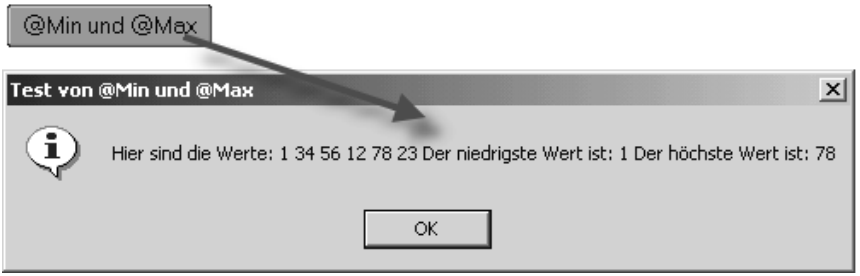
6.2.6.2 Weitere Funktionen zur Verarbeitung von Listen

Bisher war es möglich mit Hilfe der Funktionen @Min und @Max den niedrigsten oder höchsten Wert im Vergleich zweier Zahlen oder im Vergleich zweier Zahlenlisten zu ermitteln. In Version 6 ist es jetzt möglich, aus einer einzeln Zahlenliste den höchstem bzw. niedrigsten Wert zu ermitteln.

Zahl := @Min(Liste)
Zahl := @Max(Liste)

Programmtext

```
REM {Listenwerte festlegen};
Liste := 1 : 34 : 56 : 12 : 78 : 23;

REM {Listenelemente zu einer Textvariablen zusammensetzen};
ListText := „Hier sind die Werte:";
@For(i = 1; i<=@Element(Liste); i:= i + 1;
   ListText := ListText + „ „ + @Text(Liste[i]);

REM {Ausgabetext zusammensetzen};
PromptText := ListText + „ Der niedrigste Wert ist: „ +
   @Text(@Min(Liste)) + „ Der höchste Wert ist: „ +
   @Text(@Max(Liste));

REM {Text anzeigen};
@Prompt([OK]; "Test von @Min und @Max"; PromptText);
```



Die Funktionen @Min und @Max

Die Wirkung, Listen auf ihre Werte hin zu vergleichen, beschränkt sich bei den Funktionen „@Min" und „@Max" auf Listen, die Zahlen beinhalten. Zur Untersuchung von Listen mit Zeichenketten steht nun eine neue Funktion mit dem Namen „@Compare" zur Verfügung. Werden zwei Zeichenketten mit Hilfe dieser Funktion verglichen, so erhält man eine Zahl als Ergebnis. Dabei bedeutet:

- **0:** Alle Elemente der beiden Listen sind gleich.
- **-1:** Ein Element in der ersten Liste beginnt früher im Alphabet als die Elemente der zweiten Liste.
- **1:** Ein Element in der zweiten Liste beginnt früher im Alphabet als der Elemente der ersten Liste.

Die Syntax der @Compare-Funktion ist:

Zahl := @Compare(Textliste1; Textliste2; [Schlüsselwortliste])

Ist eine der beiden Listen kürzer als die andere, wird das letzte Element der kürzeren Liste mit den restlichen Elementen der längeren Liste verglichen. Folgende Schlüsselwörter können verwendet werden (mehrere Schüsselwörter werden durch „:" getrennt):

- **[CaseSensitive]** Bei dem Vergleich wird die Groß- und Kleinschreibung berücksichtigt. Das ist der Standardwert, wenn nichts angegeben wird.
- **[CaseInsensitive]** Bei dem Vergleich wird keine Groß- und Kleinschreibung berücksichtigt.
- **[AccentSensitive]** Es werden Akzente berücksichtigt. Das ist der Standardwert, wenn nichts angegeben wird.
- **[AccentInsensitive]** Es werden keine Akzente berücksichtigt.
- **[PitchSensitive]** Es werden Doppel-Byte-Zeichen berücksichtigt.
- **[PitchInsensitive]** Es werden keine Doppel-Byte-Zeichen berücksichtigt.

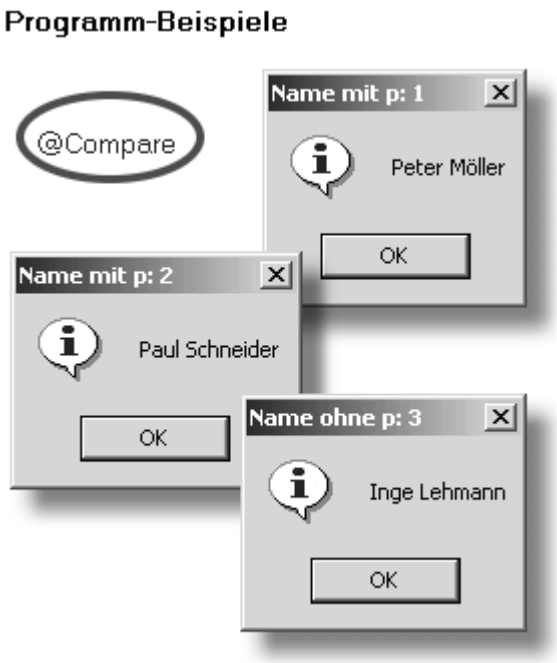
Natürlich lassen sich auf diese Weise auch einzelne Zeichenketten miteinander vergleichen.

Programmtext

```
REM {Liste erstellen};
Namen := „Peter Möller" : „Paul Schneider" : „Inge Lehmann";

REM {In der @For-Schleife werden die Namen über die @Prompt-
     Funktion angezeigt. Dabei wird geprüft, ob der einzelne Name
```

mit dem Buchstaben „p" beginnt und der Meldungstitel entsprechend angepasst wird.};

```
@For(i := 1; i <= @Elements(Namen); i:= i + 1;
@if(@Compare(Namen[i]; „p"; [CaseInsensitive]) = 1;
@Prompt([OK]; „Name mit p: „ + @Text(i); Namen[i]);
@Prompt([OK]; „Name ohne p: „ + @Text(i); Namen[i])));

REM {Das Ergebnis hat folgendes Aussehen: };
```



Die Compare-Funktion

Häufig soll eine Liste so angezeigt werden, dass die Werte in einer bestimmten Reihenfolge zu sehen sind. Diese Anzeige ist über die neue Funktion „@Sort" möglich. Die Funktion erhält eine Liste, ein Schlüsselwort und eventuell eine Formel als Parameter. Die Syntax lautet:

@Sort(Liste; [Schlüsselwort]; Formel für CustomSort)

Bei den Schlüsselwörtern kommen die gleichen Begriffe wie bei der @Compare-Funktion zum Einsatz. Zusätzlich können folgende Schlüsselwörter verwendet werden:

- **[Ascending]** aufsteigende Sortierfolge.
- **[Descending]** absteigende Sortierfolge.
- **[CustomSort]** Bei dem Schlüsselwort [CustomSort] muss zusätzlich eine Formel angegeben werden, die die Sortierfolge regelt. Mit Hilfe der Standardnamen „$A" und „$B" wird beschrieben, wie die Listenwerten verglichen und einsortiert werden sollen. Das Ergebnis der Formel muss eine Zahl ergeben. Ein Beispiel für eine solche Formel ist:

  @if(@Length($A) < @Length($B); -1; @Length($A) > @Length($B); 1; 0

Um eine Textliste zu sortieren, wird z.B. mit den Schlüsselwörtern „[Ascending]" und „[CaseInsensitive]" gearbeitet. Die Funktion hat dann die Form:

@Sort(Liste; [Ascending]:[CaseInsesitive])

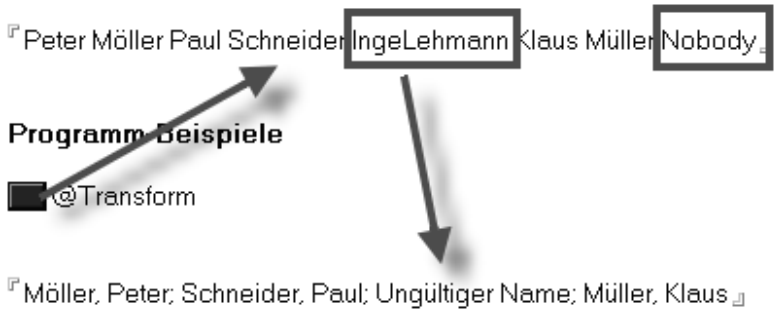
Zur Veränderung der Elemente einer Liste kann die neue Funktion „@Transform" eingesetzt werden. Mit Hilfe einer Formel werden alle Elemente einer Liste verändert und in eine neue Liste ausgegeben. Soll ein Wert nicht in der neuen Liste erscheinen, kann die neue Funktion „@Nothing" verwendet werden. Die Syntax von @Transform lautet:

NeueListe := @Transform(Liste; "Variable"; Formel)

Die Variable dient dabei als Zwischenvariable für jedes einzelne Element in der Liste bei der Formulierung in der Formel.

Programmtext

```
REM {Feld mit Startliste füllen};
FIELD Namen := „Peter Möller" : „Paul Schneider" : „Inge Lehmann" :
      „Klaus Müller" : „Nobody";

REM {Listenwerte ändern und in ein neues Feld ausgeben};
FIELD ChgNamen := @Transform(Namen; "var";
      @if(@Contains(var; „ „); @Right(var; „ „) + „„ „ +
      @Left(var; „ „); @Contains(var; „Nobody");
      @Nothing; „Ungültiger Name"))
```



@Transform und @Nothing

6.2.6.3 Feldwerte und -namen ermitteln

Eine wichtige Einschränkung der Formelsprache ist es, dass Feldnamen häufig fest in der Formel hinterlegt werden müssen. Mit Release 6 bietet die Formelsprache Funktionen zur Ermittlung des Namens und des Inhaltes des aktuellen Feldes. Darüber hinaus wird es möglich über die Funktion „@GetField" Feldnamen zu berechnen und dann auf diese Felder zuzugreifen. Die Syntax der Funktionen lautet:

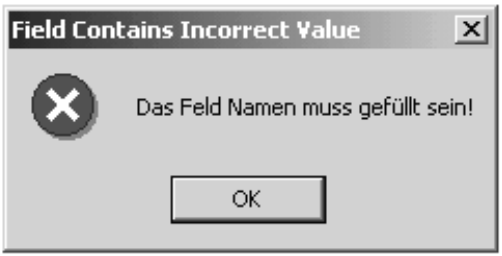
Feldname := @ThisName
Feldinhalt := @ThisValue
Feldinhalt := @GetField(Feldname als Variable oder Wert)

Formeln zur Prüfung des Feldinhaltes können mit diesen Funktionen erstellt und ohne Änderung in andere Felder mit der gleichen Prüflogik kopiert werden.

Programmtext

```
REM {Validierungsformel im Feld Namen}
@if(@ThisValue = ""; @Failure("Das Feld " + ThisName +
      „ muss gefüllt sein!"); @Success);

REM {Diese Formel kann in jedes Feld kopiert werden,
      das nicht leer sein darf.};
```

Programm-Beispiele



@ThisName und @ThisValue

Diese Seite ist zur Zeit nicht belegt.

6.3 LotusScript

Wer Basic als Programmiersprache bereits kennt, für den ist LotusScript sicher erste Wahl, denn es handelt sich hier um eine Basic-Version. Aber auch dann, wenn man andere im Designer verfügbare Programmiersprachen vorzieht, kommt man in manchen Bereichen um LotusScript nicht herum. Die Sprache ist sehr gut in die Domino-Umgebung integriert. Mit Release 5 wurde sie als LotusScript 4 mit vielen Erweiterungen implementiert. Lotus versucht offensichtlich, alle Elemente der Formelsprache auch in LotusScript zur Verfügung zu stellen. Darüber hinaus wurden eigene Objekte und Klassen zur Bearbeitung typischer Notes-Objekte in die Sprache eingefügt.

Zur Zeit arbeitet Lotus an einer neuen Entwicklungsumgebung mit dem Namen EBS (Enterprise Solution Builder), die LotusScript auch außerhalb von Domino zum Einsatz bringen soll. Bereits heute ist LotusScript auch in anderen Lotus-Produkten verfügbar (Word Pro, Freelance Graphics, Approach, Lotus 1-2-3).

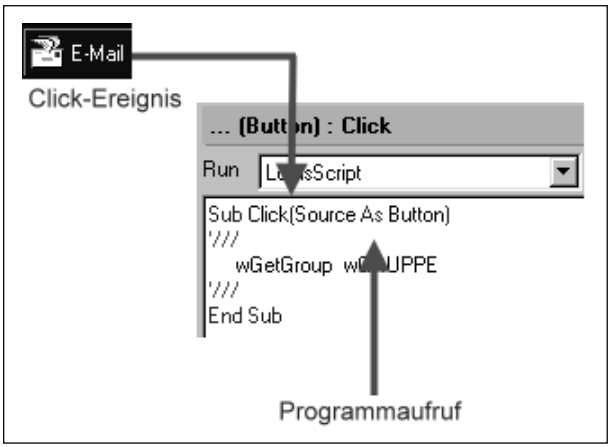
Diese Seite ist zur Zeit nicht belegt.

6.3.1 Grundlagen

Als Script-Sprache dient LotusScript dazu, bestehende Programme um zusätzliche Funktionen zu erweitern. Es ist nicht möglich, eigenständige LotusScript-Programme zu erstellen. Die Module sind vielmehr an Objekte und Ereignisse gebunden, die von den Hauptprogrammen verwendet werden.

Ein LotusScript-Programm besteht aus Definitions- und Befehlszeilen, die im Normalfall durch einen Zeilenwechsel voneinander getrennt sind. Der Text wird im Rahmen der Entwicklungsumgebung kompiliert und beim Aufruf innerhalb des Hauptprogrammes interpretiert.

Das Programm wird durch ein Ereignis aufgerufen, das Bestandteil eines Objektes im Rahmen des Hauptprogrammes ist. Eine Schaltfläche in einer Aktionsleiste verfügt z.B. über ein „Click“-Ereignis. Anklicken der Schaltfläche löst das „Click“-Ereignis aus, sodass der zu diesem Ereignis gespeicherte Programmteil ausgeführt wird.



Aufruf eines LotusScript-Programmes

6.3.1.1 Die Syntax

Eine Anweisung in LotusScript besteht aus einer Textzeile. Es ist kein spezielles Trennzeichen wie in C oder Java erforderlich. Das Zeilentrennzeichen trennt auch die Anweisungen. Soll eine Anweisung über mehrere Zeilen gehen, so wird sie mit einem Leerzeichen und einem Tiefstrich („ _ „) beendet. Die Anweisung kann dann in der nächsten Zeile fortgesetzt werden. Umgekehrt ist es auch möglich, mehrere Anweisungen in eine Zeile zu schreiben. In diesem Fall wird ein Doppelpunkt als Trennzeichen für die Anweisungen verwendet.

Beispiel für eine mehrzeilige Anweisung:

```
if Adressfeld.Values(0) = _
uidocCur.Document.Name(0) _
then goto ErrorLabel
```

Beispiel für mehrere Anweisungen in einer Zeile:

```
a = b + 1: x = a ^ 3
```

Leerzeilen im Text werden vom Compiler ignoriert und eignen sich deshalb hervorragend zur Gliederung langer Programm-Module. Zwischen den einzelnen Worten können beliebig viele Leerzeichen oder Tab-Schaltungen eingesetzt werden. Ein Leerzeichen wird als Mindestabstand verlangt.

Sprungmarken, die von Goto-Befehlen verwendet werden, sind Begriffe, die am Beginn einer Zeile stehen und mit einem Doppelpunkt abgeschlossen werden:

```
ErrorLabel:
```

LotusScript verwendet als Sprachelemente Schlüsselwörter, die der Anweisung ihre Bedeutung verleihen. Es handelt sich dabei um Befehle, Funktionen, Operatoren und Definitionen. Sie sind die eigentlichen Elemente der Script-Sprache.

Beispiele für Schlüsselwörter:

```
Abs, Dim, If, End, IsArray, Next, Public, Select
```

Zur Definition von Daten verwendet LotusScript Namen und identifiziert damit Variablen, Konstanten, Klassen, Funktionen und Prozeduren. Für solche Namen gilt:

- Namen müssen mit einem Buchstaben beginnen,
- die restlichen Zeichen können Buchstaben, Zahlen oder ein Tiefstrich sein.

LotusScript unterscheidet keine Groß- und Kleinbuchstaben. Für die Lesbarkeit eines Programmes ist es aber wesentlich günstiger, mit Groß- und Kleinbuchstaben zu arbeiten. In vielen Programmiersprachen werden die Worte durch den Einsatz von Großbuchstaben lesbarer gemacht (z. B. NotesUiDocument).

- Der ANSI-Zeichensatz mit 256 Zeichen wird unterstützt,
- die maximale Länge eines Namens beträgt 40 Zeichen.

Der Lesbarkeit dient auch das Einrücken des Programmcodes bei Blockanweisungen:

```
For i = 1 To 34
      A = A + 1
Next
```

Da Tab-Sprünge vom Compiler ignoriert werden, bieten sie sich für diese Vorgehensweise an. Eine Vorschrift für die Verwendung dieser Methode gibt es jedoch nicht.

Kommentarzeilen können durch das Schlüsselwort „REM“ oder das Hochkommazeichen (,) gekennzeichnet werden. Mehrere Zeilen werden durch %REM am Anfang und %END REM am Ende als Kommentar definiert:

```
REM Dieser Text wird von Compiler als Kommentar
REM erkannt
' Auch dieser Text ist ein Kommentar
Dim x as String      ' Mit Hochkomma kann
                     ' Kommentar am Ende
                     ' einer Zeile stehen

%REM
Dieser Text wird ebenfalls als
Kommentar erkannt
%END REM
```

6.3.1.2 Konstanten, Variablen und ihre Datentypen

Basic ist eine der Programmiersprachen, die es erlaubt, Definitionen implizit (also im laufenden Quelltext) vorzunehmen. Da diese Methode mehr zur Verwirrung als zur Übersichtlichkeit beiträgt, sollten Variablen und Konstanten grundsätzlich zu Beginn eines Programmabschnittes explizit definiert werden. Dass eine Definition nicht unterbleibt, kann mit Hilfe der „Option“-Anweisung „Option Declare“ festgelegt werden. Der Compiler prüft dann, ob alle Namen im Quelltext definiert wurden und erzeugt entsprechende Fehlermeldungen.

6.3.1.2.1 Konstanten

Konstanten können im Text als Literale und über Namen als definierte Konstanten verwendet werden. Literale werden als Zeichenketten oder Zahlen direkt in die einzelnen Anweisungen eingebunden:

```
Vorname = "Peter"
Nachname = "Maier"
Hausnummer = 78
```

Literale, die eine Zeichenkette enthalten, müssen in Begrenzungszeichen eingeschlossen werden. Als Begrenzungszeichen können verwendet werden:

```
Redezeichen ("")
senkrechte Striche (|||)
geschweifte Klammern ({})
```

Sollen diese Zeichen innerhalb eines Literals verwendet werden, so werden sie zweimal hintereinander geschrieben. Soll ein Begriff innerhalb eines Literals mit solchen Zeichen eingegrenzt werden, so ist dazu ein anderes dieser Zeichen zu verwenden.

Beispiel für Literale:

```
|Da geht die Post ab|
{Schöner Wohnen}
"Peter Meier"
```

Beispiel für die Verwendung eines Begrenzungszeichens in einem Literal:

```
"Verwenden Sie den Begrenzer || für Literale"
```

Beispiel für die Verwendung einer Eingrenzung im Literal:

```
"Man nannte ihn |Ringostar|"
```

Zur Definition von Konstanten wird das Schlüsselwort „Const“ verwendet. Je nach Art des folgenden Ausdrucks wird der Konstanten automatisch ein Datentyp zugewiesen. Die Syntax der Konstantendefinition lautet:

[Public | Private] Const Name = Ausdruck

Dabei bedeuten:

- Public | Private

  Gültigkeitsbereich der Konstanten nur innerhalb oder auch außerhalb eines Moduls

- Const

  feststehendes Schlüsselwort

- Name

  Name der Konstanten (üblicherweise in Großbuchstaben)

- Ausdruck

  Literal, eine andere Konstante, eine arithmetische Operation oder eine von bestimmten LotusScript-Funktionen, die zur Kompilierungszeit ausgewertet werden können.

Beispiel für Konstante:

```
Const STERN = "*"
Const NO_FILE = "Datei nicht gefunden!"
Const N4711% = 4711
```

Es hat sich als übliche Vorgehensweise herausgebildet, Konstanten in Großbuchstaben zu schreiben. Es ist natürlich ebenso möglich, die Groß-/Kleinschreibung zu verwenden.

LotusScript enthält feste Konstanten, die nicht definiert werden müssen:

- TRUE, FALSE
  sind Bezeichnungen, die für -1 (TRUE) bzw. 0 (FALSE) stehen.
- NULL
  beschreibt den Inhalt einer nicht initialisierten Variable.
- PI
  3,14 °
- NOTHING
  ist eine nicht zugewiesene Objektreferenz.

Die einzige Möglichkeit, einer Konstante gezielt einen Datentyp zuzuweisen, besteht darin, die speziellen Typ-Endungen zu verwenden, die im Rahmen der Variable besprochen werden.

Lotus liefert zusammen mit seinen Produkten eine oder mehrere Textdateien aus, die bereits eine Fülle von Konstantendefinitionen enthalten. Diese Dateien finden sich bei Lotus Domino im Verzeichnis „Domino“ bzw. „Notes“ und können in Script-Programmen eingebunden werden.

```
'-------------------------------------------------
'     File:       LSCONST.LSS
'     Copyright   1993 Lotus Development Corporation
'
'     Description: LotusScript Global Constants.
'
'-------------------------------------------------
' Variant DataType Values
'-------------------------------------------------

Public Const V_EMPTY = 0            ' Empty variant
Public Const V_NULL = 1             ' Variant containing Null
Public Const V_INTEGER = 2          ' Integer
Public Const V_LONG = 3             ' Long integer (4 bytes)
Public Const V_SINGLE = 4           ' Single
Public Const V_DOUBLE = 5           ' Double
Public Const V_CURRENCY = 6         ' Currency
Public Const V_DATE = 7             ' Date value
Public Const V_STRING = 8           ' String
Public Const V_DISPATCH = 9         ' OLE object
```

Die Konstanten können dann im Programm an Stelle der Zahlen verwendet werden, um die Lesbarkeit des Quelltextes zu erhöhen.

6.3.1.2.2 Variable

Durch die Definition einer Variable wird einem Speicherplatz ein Name zugewiesen. Der Inhalt des Speicherplatzes kann sich während des Programmablaufes ändern. Je nach Typ der Variable wird mehr oder weniger Speicherplatz für sie reserviert. LotusScript kennt folgende Variablentypen:

- Integer

  Zahl zwischen -32.768 und 32.767
  Kennzeichen: %

- Long

  Zahl zwischen -2.147.483.648 und 2.147.483.647
  Kennzeichen: &

- Single

  Zahl zwischen -3.402823E+38 und 3.402823E+38
  Kennzeichen: !

- Double

  Zahl zwischen -1.7976931348623158E+308 und 1.7976931348623158E+308
  Kennzeichen: #

- Währung

  Zahl auf 4 Nachkommastellen gerundet zwischen 922.337.203.685.477,5807 und 922.337.203.685.477,5807
  Kennzeichen: @

- String

  Zeichenkette, die durch den verfügbaren Arbeitsspeicher begrenzt ist.
  Kennzeichen: $

Das Typkennzeichen wird an den Variablenamen angehängt und weist einer Variable einen bestimmten Datentyp zu. Alternativ dazu kann bei der Variablendefinition der Typ auch direkt angegeben werden. Die Definitionsanweisung für Variablen lautet:

[Dim | Public | Private | Static] Name [as Typ]

Dabei bedeutet:

- Dim

  Schlüsselwort für die Variablendefinition (entspricht Private)

- Public, Private

  Gültigkeitsbereich (innerhalb einer Funktion / Prozedur, auf Modulebene usw.)

- Static

  Obwohl die Definition innerhalb einer Funktion bzw. Prozedur definiert ist, behält sie ihren Wert, wenn die Funktion / Prozedur verlassen wird.

- Name

  Name der Variable

- as Typ

  Wenn der Variablentyp angegeben wird, muss das Schlüsselwort “as” verwendet werden.

Beispiele für Variablendefinitionen:

```
Dim Test as interger
Dim i%
Public Wichtig$
```

Eine besondere Form der Variable stellt der Datentyp „Variant“ dar. Dieser Typ kann alle Datentypen aufnehmen. Intern wird verwaltet, welchen Typ die Daten wirklich haben.

Neben den einfachen Variablen können noch Zusammenfassungen mehrerer Variablen definiert werden. Dazu gehören:

- Array

  Datenelement, das mehrere Variablen eines Datentyps aufnehmen kann. Die Elemente einer Array können über ihren Index angesprochen werden.

- Variant

  Eine Variable vom Typ „Variant“ kann ebenfalls mehrere Variablen eines Typs aufnehmen.

- Liste

  Bei der Liste erhält jedes Element einen eigenen Namen.

- Benutzertyp

  Zusammenfassung von Variablen unterschiedlicher Typen.

- Klasse

  Definition eines Objektes mit Eigenschaften und Methoden.

6.3.1.2.2.1 Zahlen

Die 1Zahlentypen unterscheiden sich im Wesentlichen durch Speicherplatz, der bei ihrer Definition im Arbeitsspeicher reserviert wird. Je mehr Speicherplatz kostengünstig zur Verfügung steht, desto unwichtiger wird dieser Aspekt. Der zweite Unterschied besteht darin, ob Nachkommastellen verarbeitet werden oder nicht. Bei den Datentypen „Integer“ und „Long“ werden keine Nachkommastellen verwaltet (es handelt sich nicht um so genannte Fließkommazahlen).

Beispiele für Zahlen:

```
Dim x as Integer, v as Long
Dim w as Single, s as Double
Dim t as Currency

Dim a%
Dim b&
Dim c!, d#, e@

x = 17
v = 7
a = x - v

d = v * x
```

6.3.1.2.2.2 Zeichenketten (Datentyp String)

Unter einer Zeichenkette versteht man einen Text, der aus Buchstaben, Zahlen und Sonderzeichen bestehen kann. Eine solche Zeichenkette kann in der Länge variabel oder fest definiert werden. Eine String-Variable fester Länge erhält als Ergänzung zur Typangabe „String“ noch einen Stern und die Länge der Zeichenkette. Während bei variabler Länge nur ein Leerstring angelegt und der benutzte Speicher dynamisch angepasst wird, erhält die feste Zeichenkette eine unveränderbare Anzahl an Bytes im Speicher zugewiesen und wird mit chr(0) je Zeichen initialisiert.

Beispiele für den Datentyp String:

```
Dim Name as String
Dim Vorname$
Dim Nachname as String*20

Nachname = "Diese Zeichenkette enthält mehr" & _
"Zeichen, als die Variable aufnehmen kann."

Print Nachname

' Ausgabe:
' Diese Zeichenkette e
```

6.3.1.2.2.3 Datentyp Variant

Eine Variant-Variable kann unterschiedliche Datentypen speichern. Sie wird ohne Angabe eines Typs oder mit „as Variant“ definiert:

```
Dim Variable              ' als Variant
Dim Variable as Variant   ' ebenfalss als Variant
```

Folgende Typen können in einer Variablen vom Typ Variant enthalten sein:

- Integer
- Long
- Double
- Currency
- String
- Boolean (TRUE oder FALSE bzw. -1 oder 0)
- Datum / Zeit

- Array
- Liste
- Objektreferenz

Ist die Variant-Variable leer, so wird sie mit NULL bzw. EMPTY initialisiert. Diese Werte können mit speziellen Funktionen abgefragt werden (IsNull bzw. IsEmpty).

Beispiel Variant:

```
Dim a as Variant

a = "Hugo"                ' a enthält einen String
a = 12345                 ' a enthält eine Zahl
                          ' vom Typ Integer
```

6.3.1.2.2.4 Arrays

In Arrays lassen sich mehrere Datenelemente zusammenfassen. Die Elemente erhalten einen Positionszähler (Index), der bei 0, bei 1 oder einer definierten Zahl beginnen kann. Ob die Zählung bei 0 oder bei 1 beginnt, kann über die Anweisung «Option Base» gesteuert werden. Ohne diese Anweisung wird bei 0 begonnen. LotusScript unterscheidet zwischen festen Arrays, deren genaue Länge bei der Definition festgelegt wird, und solchen mit dynamischer Länge. Die Länge der dynamischen Arrays wird über die Anweisung «Redim» festgelegt. Sollen die bisherigen Werte der Arrays bei «Redim» bestehen bleiben, so kommt das Schlüsselwort «Preserve» zum Einsatz. Arrays können immer nur einen Datentyp enthalten. Der Datentyp wird bei der Definition angegeben. Arrays können mehrdimensional sein. Bis zu 8 Dimensionen sind möglich.

Beispiele:

```
REM Mit Option Base 1 wird festgelegt, dass der
REM Index bei 1 beginnt
REM Die Anzahl der Elemente beträgt 5

Option Base 1

Dim Abteilungen(5) as String

' Ohne Option Base kann die gleiche Wirkung
' erzielt werden, wenn die Unter- und Obergrenze
' der Array angegeben werden

Dim Abteilungen(1 To 5) as String

' Eine einfache Definition ohne Option Base
' verteilt die Elemente auf 0 bis 4

Dim Abteilungen(5)

REM Dynamische Array:

Opition Base 1
Dim Abteilungen() as String

REM weitere Anweisungen

Redim Abteilungen(4) as String
Abteilungen(1) = "Marketing"
Abteilungen(2) = "Buchhaltung"
Abteilungen(3) = "Versand"
Abteilungen(4) = "Lager"

Redim Preserve Abteilungen(5) as String
Abteilungen(5) = "Geschäftsleitung"

' Mehrdimensionale Array:

Dim Abteilungen(1 To 5, 1 To 3)
```

6.3.1.2.2.5 Listen

Wie die Array kann eine Liste mehrere Elemente aufnehmen. Sie verwendet zur Kennzeichnung keine Zähler, sondern sog. „List-Tags“ als eindeutige Namen je Element. Außerdem ist nur eine Dimension möglich. Alle Elemente müssen vom gleichen Datentyp sein. Die Größe der Liste kann dynamisch durch Hinzufügen oder Löschen von Elementen verändert werden.

Beispiel:

```
Dim Adresse List as String

Adresse("Name") = "Peter Müller"
Adresse("Anschrift") = "Hauptstrasse 34"
Adresse("Plz") = "12345"
```

6.3.1.2.2.6 Benutzerdefinierte Datentypen

Mit Hilfe der Anweisung „Type“ lassen sich Variablen mit unterschiedlichen Datentypen zusammenfassen und unter einem übergeordneten Namen verwenden. Ist ein Typ definiert, so kann er zur Definition von Variablen eingesetzt werden. Die Elemente der neuen Variablen werden angesprochen, indem der Name der Variablen um einen Punkt und den Namen des Elementes ergänzt wird.

Beispiel:

```
Type Adresse
      Name as String
      Anschrift as String
      Plz as Long
End Type

Dim Firma as Adresse

Firma.Name = "Softwareschmiede AG"
Firma.Anschrift = "Im Park 56"
Firma.Plz = 34256
```

Bei der Definition der Typ-Elemente wird das Schlüsselwort „Dim“ nicht verwendet.

6.3.1.2.2.7 Klassen und Objekte

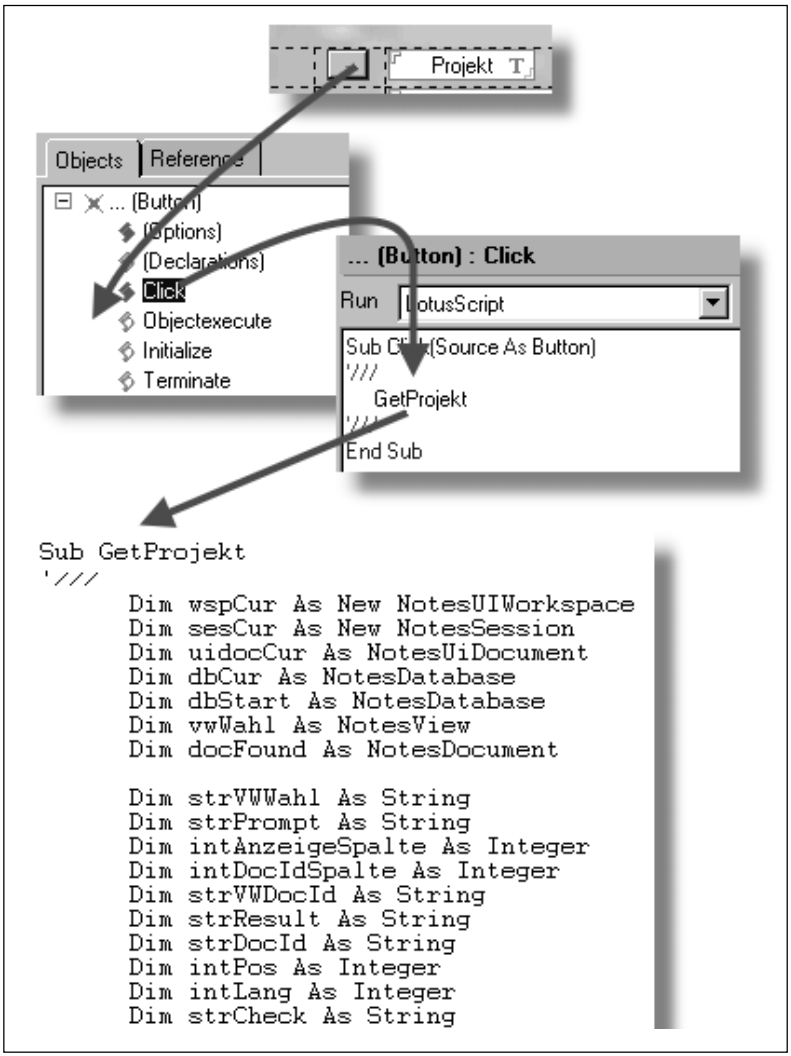
LotusScript unterstützt und verwendet Klassen und Objekte sehr intensiv. Die verschiedenen Lotus-Anwendungen stellen eigene Klassen als Erweiterung zu LotusScript zur Verfügung. Auch der Benutzer kann eigene Klassen und Objekte erstellen. Die Definition und Verwendung dieser Sprachelemente werden in einem späteren Kapitel dieses Handbuches vorgenommen.

6.3.1.2.3 Prozeduren und Funktionen

Ein Programm-Modul gehört zu einem Objekt und seinen Ereignissen. Es enthält bestimmte Standardelemente, die von der Art des Objektes abhängen. Das Objekt „Button“ z.B. enthält die Ereignisse:

- Options
- Declarations
- Click
- Objectexecute
- Initialize
- Terminate

Jedem dieser Ereignisse kann ein Programmcode zugeordnet werden. Für das „Click“-Ereignis sieht das dann folgendermaßen aus:



Das Programm-Modul zum Button-Object

Das „Click“-Ereignis enthält die Prozedur „Click“. Diese Prozedur erhält einen Parameter (Source As Button) und ruft ihrerseits eine neue Prozedur mit dem Namen „GetProject“ auf.

Diese Unterteilung des Programmcodes in Prozeduren erhöht die Lesbarkeit, erleichtert die Programmpflege und erzeugt wieder verwendbare Bausteine.

LotusScript unterscheidet zwischen Prozeduren und Funktionen. Funktionen können einen Wert an das aufrufende Modul zurück liefern, Prozeduren nicht. Die Anweisungen von LotusScript stellen Prozeduren und Funktionen dar. Sie können durch benutzereigene ergänzt werden.

Die Syntax für Prozeduren lautet:

[Public | Private] [Static] Sub Name(Parameterliste)

die der Funktion:

[Public | Private] [Static]
Function Name [(Parameterliste)][as Typ]

Dabei bedeutet:

- Public, Private

  Gültigkeitsbereich

- Static

  Obwohl die Definition innerhalb einer Funktion bzw. Prozedur definiert ist, behält sie ihren Wert, wenn die Funktion / Prozedur verlassen wird.

- Sub / Function

  Schlüsselwort

- Name

  Name der Variablen

- Parameterliste

  in Klammern eingeschlossene, durch Kommas getrennte Liste der an die Prozedur / Funktion übergebenen Parameter mit Angabe von deren Datentyp

- As Typ

  Wenn der Variablentyp für den Rückgabewert angegeben wird, muss das Schlüsselwort „As“ verwendet werden.

Die Prozedur / Funktion wird mit dem Schlüsselwort „End Sub“ bzw. „End Function“ abgeschlossen.

Parameter können „by value“ oder „by reference“ übergeben werden. „By value“ bedeutet, dass eine Kopie des Variableninhaltes übergeben wird. „By reference“ bedeutet, dass ein Zeiger auf die eigentliche Variable übergeben wird. Die Übergabe „by reference“ führt dazu, dass der Inhalt der Variable selbst verändert wird, wenn die Prozedur oder Funktion den Wert dieser Variable ändert. Diese Form der Parameterübergabe ist der Standard in LotusScript. Die Übergabe „by value“ muss ausdrücklich angegeben werden.

Beispiele:

```
Sub Kreisverkehr(Richtung as String)

End Sub

Function IsMontag(ByVal Datum as string, Count _
as Integer) as String

End Function
```

Der Aufruf einer Prozedur erfolgt innerhalb eines Programmtextes mit ihrem Namen. Sollen die Parameter in Klammern aufgeführt werden, muss sie mit „Call“ aufgerufen werden.

Beispiele:

```
RunWindow "Hallo"
Call RunWindow("Hallo")
```

Der Rückgabewert einer Funktion wird zugewiesen, indem innerhalb der Funktion dem Funktionsnamen ein Wert zugewiesen wird:

```
function cFunktion (Input as Integer) as Integer

xFunktion = 123

End Function
```

Der Aufruf kann direkt als Variablenzuweisung verwendet werden, wenn die Funktion über einen Rückgabewert verfügt.

Beispiel: `Anzahl = xFunktion(Startwert)`

Gibt die Funktion keinen Wert zurück, wird sie ohne Klammern aufgerufen.

6.3.1.3 Operatoren und Ausdrücke

Zur Durchführung von Anweisungen stellt LotusScript ein umfangreiches Repertoire von Operatoren und Ausdrucksformen zur Verfügung. Je nach Datentyp stehen unterschiedliche Operatoren bereit oder sie haben trotz gleichem Aussehen unterschiedliche Auswirkungen. Ein Operator dient der Beschreibung dessen, wie der Wert eines Elementes durch ein anderes Element verändert werden soll. Ausdrücke sind Kombinationen von Elementen und Operatoren, die zur Laufzeit einen Wert ergeben. Das Ergebnis eines Ausdrucks kann sein:

- eine Zahl
- eine Zeichenkette
- NULL
- EMPTY
- NOTHING
- True oder False

Bei den Operatoren unterscheidet LotusScript zwischen

- arithmetischen Operatoren

```
^     Potenz
-     Minus als Vorzeichen
*     Multiplikation
/     Division
\     Integer Division
Mod   Modulo (Rest einer Ganzzahlendivision)
+     Plus
-     Minus
```

- Operatoren für Zeichenketten

```
&     Verkettung bzw. Addition
```

- Vergleichsoperatoren

```
=     gleich
<>    ungleich
><    ungleich
<     kleiner
>     größer
<=    kleiner oder gleich
=<    gleich oder kleiner
>=    größer oder gleich
=>    gleich oder größer
like enthält einen Teilstring
```

- Bezug zu einem Objekt

```
Is    Objektvergleich
IsA   Ermittlung eines Objekttyps
```

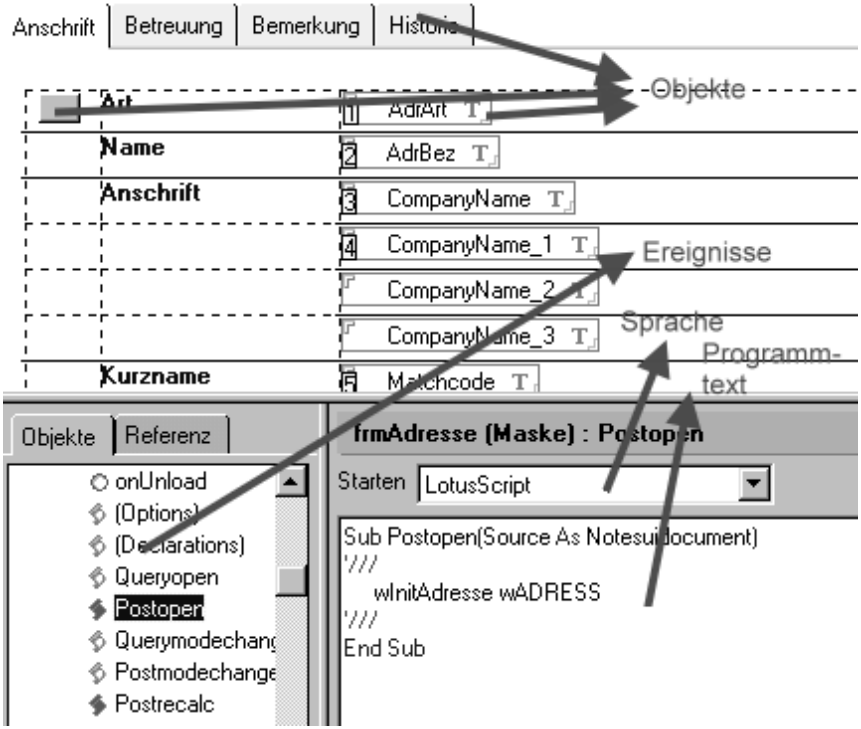
- Logische Operatoren

```
Not   Verneinung
And   logisches "Und"
Or    logisches "Oder"
Xor   logisch exclusives "Oder"
Eqv   logische Entsprechung
Imp   logische Einbeziehung
```

Die logischen Operatoren werden für die bitweise Verarbeitung entsprechend verwendet.

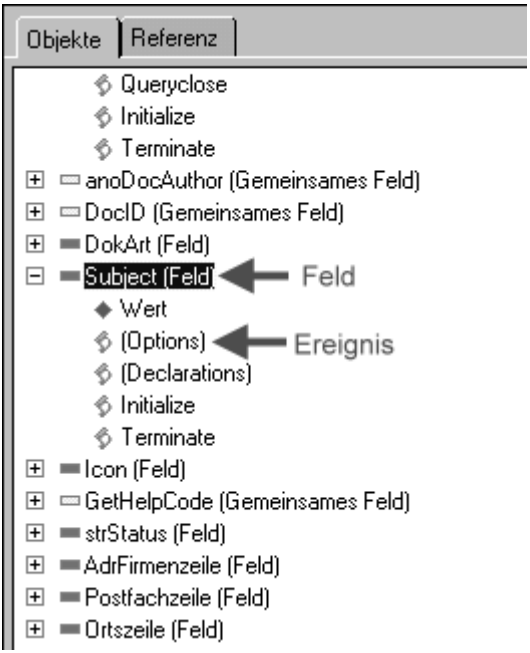
6.3.2 LotusScript im Designer

Da die Programmiersprache LotusScript voll in die Domino- und Notes-Umgebung integriert ist, kann der Programmtext jeweils dort hinterlegt werden, wo er zur Wirkung kommen soll.



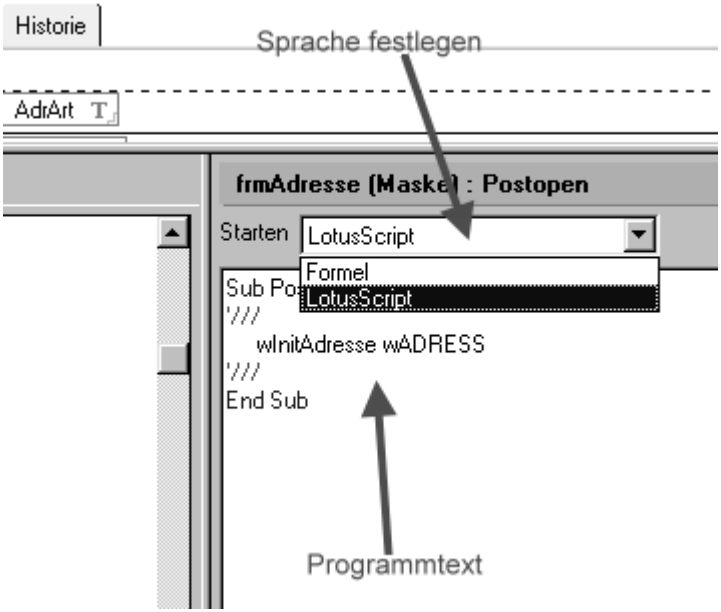
Programmtexte im Designer

Ausgehend vom Objekt (also z.B. einem Feld in einer Maske) werden die Ereignisse in der linken Spalte unter dem Register „Objekt“ angezeigt. Es finden sich dort alle aktuell verfügbaren Objekte mit den dazu gehörenden Ereignissen.



Objekte und Ereignisse

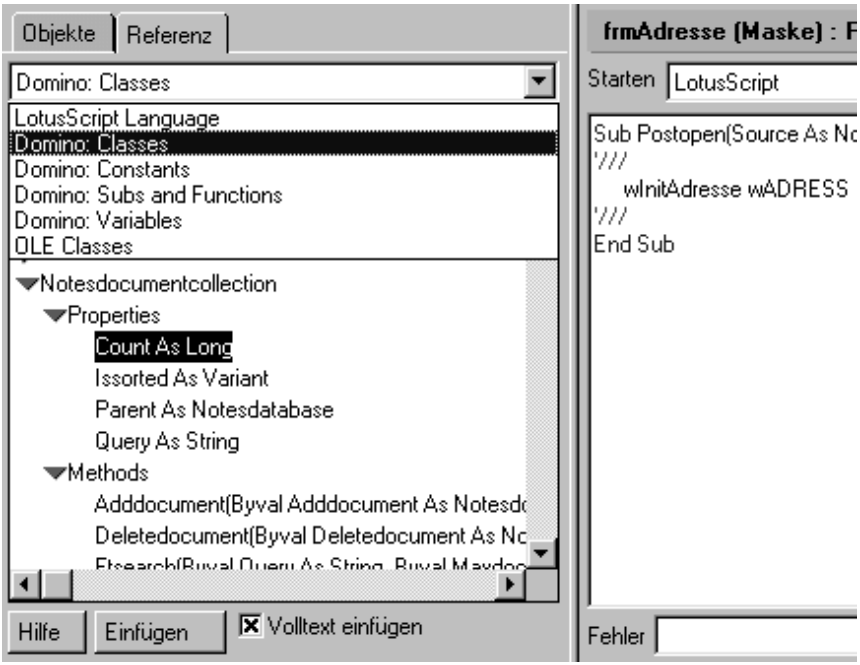
Im linken Bereich wird im Feld „Starten“ ausgewählt, welche Programmsprache verwendet werden soll. Es ist durchaus möglich, dass bei einem Ereignis nur eine Sprache verwendet werden kann.



Die Programmiersprache

Die älteren Ereignisse können meist nur in der Formelsprache, die jüngsten nur in JavaScript programmiert werden. Teilweise kann zwar ein Programmtext eingegeben werden, er wirkt aber nur unter bestimmten Voraussetzungen (z.B. mit einem bestimmten Browser zusammen).

Unter dem Register Referenz – auf der linken Seite des Designerfensters – verbirgt sich ein umfangreiches Hilfesystem, das dem Programmierer viel Arbeit und Suche erspart.



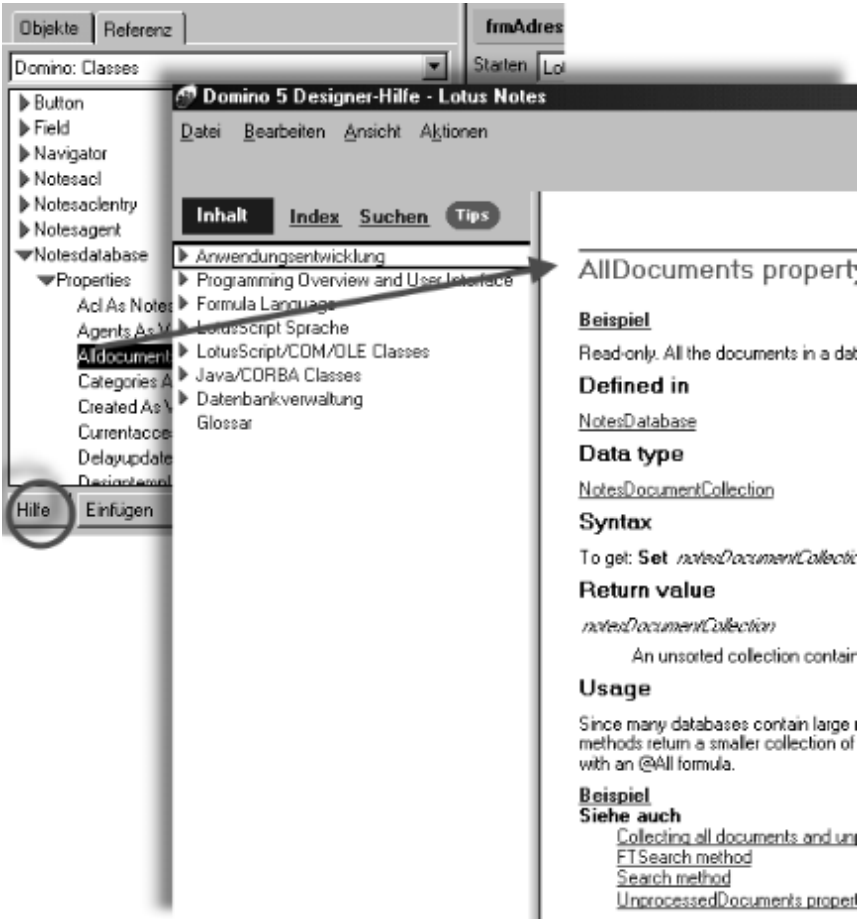
Referenz des Domino Designers

Das Auswahlfeld bietet die Positionen

- LotusScript
- Domino-Klassen
- Domino-Konstante
- Domino-Prozeduren und -Funktionen
- Domino-Variable
- OLE-Klassen

Darunter wird dann eingeblendet, was der jeweiligen Auswahl entspricht. Sobald eine Zeile ausgewählt ist, kann sie per Doppelklick in das rechte Textfenster übernommen und an der Cursorposition eingefügt werden. Je nachdem, ob das Feld “Volltext einfügen” angekreuzt ist oder nicht, wird alles oder nur der Hauptteil der Zeile eingefügt (bei einer Methode wird als Hauptteil der Name eingefügt, die Parameter nicht).

Eine wichtige Funktion ist die direkte Verbindung zum Hilfesystem. Ist die Designer-Hilfedatenbank auf dem Arbeitsplatzrechner installiert, so kann zu jedem Punkt der Referenz (mit Ausnahme der OLE-Klassen) der passende Hilfetext in einem Fenster aufgerufen werden.



Hilfe zur Referenz

Da die Hilfeseite als getrennte Anwendung gestartet wird, kann in dem Hilfefenster unter Verwendung der normalen Hilfefunktionen beliebig weitergesucht werden. Das bedingt aber auch, dass die Hilfefenster ausdrücklich geschlossen werden müssen, sonst verbleiben immer mehr solcher Hilfe-Sitzungen als eigene Windows-Fenster im Arbeitsspeicher bzw. am Bildschirm.

6.3.2.1 Der Umgang mit Zahlen

Bei der Verwendung von Zahlen richtet sich die Reihenfolge ihrer Verarbeitung in Ausdrücken nach den allgemeinen Regeln der Mathematik. Bei der Ermittlung von Rechenergebnissen ist also auf die sorgfältige Verwendung von Klammern zu achten. Die Rechenvorgänge und Datentypen hängen eng miteinander zusammen. Durch eine Division zweier Integer-Zahlen kann z.B. eine Zahl mit Nachkommastellen vom Typ Single entstehen. Die Zuweisung von Ergebnissen zu Variablen mit falschem Datentyp führt bei der Ausführung des Programms zu einer Fehlermeldung. Die Fehlermeldung kann lauten:

- Typemismatch oder
- Overflow

Je nachdem, ob ein ungültiger Typ eingesetzt wurde oder der Wertebereich des Datentyps überschritten wurde.

6.3.2.1.1 Zahlen vom Typ Integer

Integer-Zahlen haben den Gültigkeitsbereich von -32.768 bis 32767 und werden mit einer Anweisung definiert

```
Dim Zahl as Integer
Dim Zahl%
```

Welche Form der Definition man wählt, ist technisch unerheblich. Beide Versionen sind gültig. Da die Form des angehängten Kürzels in anderen Programmiersprachen weniger üblich ist und die Lesbarkeit eher mit einem Prefix erhöht wird, setzt sich immer mehr die erste Form der Definition durch. Bei Zählvariablen für Schleifen wird häufig auf das Präfix verzichtet. Lotus selbst verwendet alle Formen der Definition und kann deshalb nur schwer als Vorbild dienen. Wir verwenden in diesem Handbuch „int“ als Prefix für Integer-Variable.

Bei arthmetischen Operationen können Integer-Zahlen verwendet werden. Ergebnisse mit Nachkommastellen, die einer Integer-Variablen zugewiesen werden, rundet LotusScript automatisch zu Ganzzahlen.

Beispiel:

```
Dim intV1 as Integer
Dim intV2 as Integer
Dim intErg as Integer

intV1 = 7
intV2 = 3
intErg = intV1 / intV2

Ergebnis:
intErg enthält 2
```

Eine Anweisung, die den Wertebereich der Integerzahl sprengt, führt zu einem Fehler:

```
Dim intErg as Integer

intErg = 10000 * 100
```

6.3.2.1.2 Zahlen mit Nachkommastellen

Drei verschiedene Datentypen unterscheidet LotusScript bei den Zahlen mit Nachkommastellen:

- Single
  Speicherbedarf 4 Bytes
- Double
  Speicherbedarf 8 Bytes
- Currency
  Speicherbedarf 8 Bytes

Der Datentyp Currency unterscheidet sich dadurch, dass er eine feste Anzahl von vier Nachkommastellen hat. Er ist für die Verwendung bei Währungsbeträgen gedacht.

6.3.2.1.3 Funktionen für Zahlen

Lotus Script enthält eine Reihe von Funktionen zur Manipulation von Zahlen, soweit das nicht bereits mit den normalen Operatoren erfolgen kann:

- Abs
  liefert den absoluten Wert einer Zahl.
- ACos, ASin, ATn, ATn2, Cos, Sin, Tan
  Winkelfunktionen
- Bin
  gibt den Binärwert einer Zahl als Zeichenkette zurück.
- Exp
  berechnet e (die Basis des natürlichen Logarithmus) potenziert mit der angegebenen Zahl.
- Fix
  ermittelt den Integerwert einer Zahl.
- Format
  Die Format-Funktion ermöglicht es, über eine Schablone das Aussehen einer Zahl zu definieren und ihr das Aussehen der Schablone zuzuweisen.
- Fraction
  gibt den Nachkommastellen-Anteil einer Zahl zurück.
- Hex
  liefert den Hexadezimalwert einer Zahl.

- Int
  ermittelt den ganzzahligen Wert, der der angegebenen Zahl an nächsten liegt.

- IsNumeric
  prüft, ob ein Ausdruck eine Zahl ergibt.

- Len
  liefert die Länge einer Zahl in Anzahl der belegten Bytes.

- Log
  liefert den natürlichen Logarithmus einer Zahl.

- Oct
  gibt den Oktalwert einer Zahl zurück.

- Rnd
  generiert eine Zufallszahl zwischen 0 und 1.

- Round
  rundet eine Zahl auf die angegebene Anzahl von Nachkommastellen.

- Sgn
  ermittelt das Vorzeichen einer Zahl.

- Sqr
  berechnet die Quadratwurzel einer Zahl.

Beispiele:

```
' Zahlen definieren
Dim x As Integer
Dim y As Double
Dim z As Currency
Dim strX As String
Dim strY As String

' Werte zuweisen
x = 23
y = 2.890

'Multiplikation
z = x * y

' Division
x = z / y

' Zahl als Zeichenkette
strX = "5"
```

```
' Division durch String
y = z / strX

' Fraction liefert Nachkommastellen
yy = Fraction(y)

' Int wandelt in Integer um
xx = Int(y)

' Addition
strX = 5 + 5

' Verkettung
strY = 5 & 5

' Zeichenkette als Zahl ergibt
' Fehler: Typemismatch beim Compilieren
' z = "Hallo"

' Integerwert über 32767
' Fehler: Overflow
x = 4556 * strY
```

Fehler Overflow

```
' Ergebnisse
Msgbox "z = " & Str$(z) & Chr$(10) & _
"x = " & Str$(x) & Chr$(10) & _
"y = " & Str$(y) & Chr$(10) & _
"yy = " & Str$(yy) & Chr$(10) & _
"xx = " & Str$(xx) & Chr$(10) & _
"strX = " & strX & Chr$(10) & _
"strY = " & strY
```

Ergebnisse der Arbeit mit Zahlen

6.3.2.2 Arbeiten mit Zeichen und Zeichenketten

Zeichenketten erhalten den Datentyp „String“. Sie werden definiert mit Typangabe oder mit dem Suffix „$“. Zeichenketten können mit einer festen oder auch variablen Länge definiert werden.

Beispiele:

```
Dim strName as String      ' Zeichenkette mit
                           ' variabler Länge und
                           ' Typ
Dim strName$               ' Suffix "$"
Dim strName$ * 10          ' feste Länge
```

Wird einer Zeichenkette mit fester Länge ein Wert zugewiesen, der ihre Länge übersteigt, so geht der überschüssige Teil verloren.

Zeichenketten können mit Hilfe des „&“-Zeichens miteinander verbunden werden.

Beispiele:

```
Dim strVorname as String
Dim strName as String
Dim strFullName as String*20

strVorname = "Hans Peter "
strName = "von Müller-Vorweg"
strFullName = strVorname & strName

Ergebnis:
strFullName enthält "Hans Peter von Müller"
```

6.3.2.2.1 Funktionen zur Bearbeitung von Zeichenketten

Für die Bearbeitung von Zeichenketten sind in LotusScript mehr Funktionen verfügbar als in anderen Basic-Dialekten. Das rührt daher, dass in der Formelsprache spezielle Funktionen enthalten sind, die jetzt auch in LotusScript übernommen wurden.

- Format
  Die Format-Funktion ermöglicht es, über eine Schablone das Aussehen einer Zeichenkette zu definieren und ihr dieses Aussehen zuzuweisen.

- FullTrim
  entfernt Leerzeichen am Anfang und am Ende einer Zeichenkette sowie doppelte Leerzeichen innerhalb einer Zeichenkette.

- InStr, InStrB, InstrBP, InStrC
  geben die Position eines angegebenen Zeichens in einem String zurück.

- LCase
  wandelt alle Buchstaben einer Zeichenkette in Kleinbuchstaben um.

- Left$, LeftB$, LeftBP$, LeftC$
  ermitteln einen Teil einer Zeichenkette in der angegebenen Länge von links nach rechts.

- Len, LenB, LenBP, LenC
  ermitteln die Länge einer Zeichenkette. Bei Strings mit fester Länge wird immer die definierte Länge zurückgegeben.

- LTrim$
  entfernt führende Leerzeichen in einer Zeichenkette.

- Mid$, MidB$, MidBP$, MidC$
  geben einen Teil-String aus einer angegebenen Zeichenkette ab der festgelegten Position und in der genannten Länge zurück oder ersetzen einen Teil einer Zeichenkette durch eine andere.

- Right$, RightB$, RightBP$, RightC$
  ermitteln einen Teil einer Zeichenkette in der angegebenen Länge von rechts nach links.

- RSet
  übernimmt eine Zeichenkette in einen String mit fester Länge und richtet die Zeichenkette nach rechts aus.

- RTrim
  entfernt alle Leerzeichen am Ende einer Zeichenkette.

- Space
  liefert die angegebene Anzahl von Leerzeichen als String.

- StrCompare
  vergleicht zwei Zeichenketten und gibt das Ergebnis in Abhängigkeit von der angegebenen Methode zurück.

- StrConv
  wandelt eine Zeichenkette mit Hilfe einer angegebenen Methode um.

- StrLeft
  gibt alle Zeichen zurück, die links von einem Such-String in einer Zeichenkette stehen. Dabei kann angegeben werden, wie oft der Such-String gefunden werden muss, damit die Ausgabe wirksam werden soll. Die Suche erfolgt von links nach rechts.

- StrLeftBack
  Es werden alle Zeichen zurückgegeben, die links von einem Such-String in einer Zeichenkette stehen. Dabei kann angegeben werden, wie oft der Such-String gefunden werden muss, damit die Ausgabe wirksam werden soll. Die Suche erfolgt von rechts nach links.

- StrRight
  gibt alle Zeichen zurück, die rechts von einem Such-String in einer Zeichenkette stehen. Dabei kann angegeben werden, wie oft der Such-String gefunden werden muss, damit die Ausgabe wirksam werden soll. Die Suche erfolgt von links nach rechts.

- StrRightBack
  gibt alle Zeichen zurück, die rechts von einem Such-String in einer Zeichenkette stehen. Dabei kann angegeben werden, wie oft der Such-String gefunden werden muss, damit die Ausgabe wirksam werden soll. Die Suche erfolgt von rechts nach links.

- String$
  erstellt eine Zeichenkette aus dem angegebenen Zeichen in der angegebenen Länge.

- Trim$
  entfernt alle Leerzeichen am Anfang und am Ende einer Zeichenkette.

- UCase
  wandelt alle Buchstaben einer Zeichenkette in Großbuchstaben um.

Beispiele:

```
-
' Variable definieren
Dim strName As String
Dim strVorName As String
Dim strNachName As String
Dim strFullName As String
Dim intPos As Integer

' Variable vorbelegen
strName = "Peter Schulze"

' Mit Instr die Position des Leerzeichens
' ermitteln
intPos = Instr(1, strName, " ")



' Wenn ein Leerzeichen gefunden wurde
' Vor- und Nachname separieren und in Form von
' Nachname, Vorname abspeichern
If intPos > 0 Then
   strVorname = Mid$(strName, 1, intPos - 1)
   strNachName = Mid$(strName, intPos + 1, _
         Len(strName) - intPos)
   strFullName = strNachName & ", " & strVorName

   ' Ergebnis anzeigen
   Msgbox strName & Chr$(10) & _
   strVorname & Chr$(10) & _
   strNachName & Chr$(10) & _
   strFullName
End If

' Mit Left$ und Right$ arbeiten
intPos = Instr(1, strName, " ")
```

```
If intPos > 0 Then
   strVorname = Left$(strName, intPos - 1)
   strNachName = Right$(strName, Len(strName) _
         - intPos)
   strFullName = strNachName & ", " & strVorName

   ' Ergebnis anzeigen
   Msgbox strName & Chr$(10) & _
   strVorname & Chr$(10) & _
   strNachName & Chr$(10) & _
   strFullName
End If

' Mit StrLeft$ und StrRight$ arbeiten
strVorname = Strleft(strName, " ")
strNachName = Strright(strName, " ")


' Wenn ein Name gefunden wurde
If strName <> "" Then
   strFullName = strNachName & ", " & strVorName

   ' Ergebnis anzeigen
   Msgbox strName & Chr$(10) & _
   strVorname & Chr$(10) & _
   strNachName & Chr$(10) & _
   strFullName
End If
```

LotusScript - Beispiele

Ergebnis der Zeichenkettenbeispiele

6.3.2.3 Datums- und Zeitwerte einsetzen

Für die Arbeit mit Datum und Zeit enthält LotusScript keine eigenen Typen. Es verwendet stattdessen den Typ Variant, der in der Lage ist alle Datentypen aufzunehmen. Innerhalb des Varianttyps speichert LotusScript das Datum und die Zeit als Zahl. Dabei enthalten die Stellen vor dem Komma das Datum und die Stellen nach dem Komma die Zeit. Es können Werte zwischen dem 1. Januar 100 und dem 31. Dezember 9999 als Datum verwendet werden.

6.3.2.3.1 Funktionen für Datums- und Zeitwerte

Da man kaum mit den Zahlenwerten der internen Speicherung von LotusScript direkt arbeiten wird, kommen folgende Funktionen für die Arbeit mit solchen Werten zur Anwendung:

- CDat
  wandelt eine Zahl oder eine Zeichenkette in einen Datumswert um.
- Date
  liefert oder setzt das Systemdatum.
- DateNumber
  wandelt Jahr, Monat und Tag in einen Datumswert um.
- DateValue
  wandelt eine Zeichenkette in einen Datumswert um.
- Day
  ermittelt den Tag aus einem Datums-/Zeitwert.

- Format
  regelt die Ausgabeform eines Zeit-/Datumswertes.
- Hour
  gibt die Stunde aus einem Datums-/Zeitwert zurück.
- IsDate
  prüft, ob eine Variantvariable einen Zeit-/Datumswert enthält.
- Minute
  liefert die Minuten aus einem Datums/Zeitwert.
- Month
  ermittelt den Monat aus einem Zeit-/Datumswert.
- Now
  gibt das aktuelle Systemdatum und die Zeit zurück.
- Second
  ermittelt die Sekunden aus einem Datums-/Zeitwert.
- Time
  liefert oder setzt die Systemzeit.
- TimeNumber
  wandelt Stunden, Minuten und Sekunden in eine Zahl in einer Variablen vom Typ Variant um.
- Timer
  Anzahl Sekunden seit Mitternacht.
- TimeValue
  konvertiert eine Zeichenkette in einen Zeitwert.

- Today
  entspricht “Date”.
- Weekday
  Nummer des Tages (seit Sonntag) aus einem Datums-/Zeitwert.
- Year
  ermittelt das Jahr aus einem Datums-/Zeitwert.

6.3.2.4 Der Datentyp Variant

Variable vom Datentyp Variant können andere Datentypen aufnehmen:

- Zahlen
- Boolsche Variable (Wahr oder Falsch)
- NULL (Nichts)
- EMPTY (Leer)
- Datums-/Zeitwerte

Sie werden über die Dim-Anweisung mit oder ohne Typangabe definiert.

Beispiele:

```
Dim varName as Variant
Dim varname
```

Wird eine Variable als Variant definiert, erhält sie den Wert EMPTY. Da der Datentyp des Wertes einer Variantvariablen unterschiedlich sein kann, ist es auf dem Umweg über Variantvariable möglich, in einer Array oder einer Liste Werte mit unterschiedlichen Datentypen zu speichern. Da solche Variable mehr Speicherplatz verbrauchen und bei der Verwendung jeweils umgerechnet werden müssen, sind Variable vom Typ Variant sehr viel großzügiger im Verbrauch von Ressourcen. Das bedeutet mehr Speicherplatzverbrauch und langsamere Geschwindigkeit zur Laufzeit.

6.3.2.4.1 Funktionen für die Arbeit mit Variantvariablen

Die Funktionen, die zur Verarbeitung von Variantvariablen zur Verfügung gestellt werden, sind größtenteils zur Umwandlung von anderen und in andere Datentypen erforderlich.

- CCur
  Umwandlung eines numerischen oder String-Wertes in einen Wert vom Typ Währung
- CDat
  Umwandlung eines numerischen oder String-Wertes in einen Wert vom Typ Datum/Zeit
- CDbl
  Umwandlung eines numerischen oder String-Wertes in einen Wert vom Typ Double
- CInt
  Umwandlung eines numerischen oder String-Wertes in einen Wert vom Typ Integer

- CLng
  Umwandlung eines numerischen oder String-Wertes in einen Wert vom Typ Long

- CSng
  Umwandlung eines numerischen oder String-Wertes in einen Wert vom Typ Single

- CStr
  Umwandlung eines numerischen Wertes in einen Wert vom Typ String

- CVar
  wandelt einen Ausdruck in eine Variantvariable um.

- FullTrim
  entfernt Leerzeichen am Anfang und am Ende einer Zeichenkette und doppelte Leerzeichen innerhalb einer Zeichenkette in Variantvariablen.

- IsEmpty
  Ist eine Variantvariable ohne Inhalt?

- Left, LeftB, LeftBP, LeftC
  ermitteln einen Teil einer Zeichenkette in der angegebenen Länge von links nach rechts und geben eine Variantvariable zurück.

- Mid, MidB, MidBP, MidC
  ermitteln den Teil einer Zeichenkette oder ersetzen einen Teil einer Zeichenkette und liefern eine Variantvariable zurück.

- Right, RightB, RightBP, RightC
  ermitteln einen Teil einer Zeichenkette in der angegebenen Länge von rechts nach links und liefern eine Variantvariable zurück.

- RTrim
  entfernt alle Leerzeichen am Ende einer Zeichenkette in einer Variantvariablen.

- Trim
  entfernt alle Leerzeichen am Anfang und am Ende einer Zeichenkette in einer Variantvariablen.

6.3.2.5 Arrays

Der Einsatz von Arrays wird in LotusScript sowohl mit ein- als auch mit mehrdimensionalen Arrays unterstützt. Bis zu acht8 Dimensionen kann ein Array haben. Die Elemente eines Arrays sind vom gleichen Typ und können über eine Zählvariable, den Index, und den Namen des Arrays angesprochen werden. Wird die Anzahl der Elemente bereits bei der Definition des Arrays festgelegt, so spricht man von einem statischen Array. Wird sie erst später angegeben, so ist sie variabel und man spricht von einer dynamischen Array. Der Index ist eine Integer-Variable und die Anzahl der Elemente muss sich deshalb in deren Wertebereich halten.

Beispiele:

```
Dim ArrA(3) as String
Dim ArrB(1 To 3) as Integer

ArrA(0) = "Hallo"
ArrA(1) = "Guten Tag"
ArrA(2) = "Guten Morgen"

ArrB(1) = 7
ArrB(2) = 11
ArrB(3) = 21


Dim ArrA(2, 3) as Double
Dim ArrB(3 To 7, 4 To 67) as String

ArrA(0, 0) = "Hallo"
ArrA(0, 1) = "Guten Tag"
ArrA(0, 2) = "Guten Morgen"
ArrA(1, 0) = "Tschüß"
ArrA(1, 1) = "Ade"
ArrA(1, 2) = "Auf Wiedersehen"
```

Die Definition des Arrays hat folgende Elemente:

Dim [Public | Private | Static] Name (Grenzen) As Typ

Dabei bedeuten:

Dim
Schlüsselwort für die Arraydefinition

Public, Private
Gültigkeitsbereich (innerhalb einer Funktion / Prozedur, auf Modulebene usw.)

Static
Obwohl die Definition innerhalb einer Funktion bzw. Prozedur definiert ist, behält das Array seine Werte, wenn die Funktion / Prozedur verlassen wird.

Name
Name des Arrays

Grenzen
Anzahl der Elemente je Dimension oder Von-/Bis-Werte je Dimension oder keine Angabe (dynamisches Array)

As Typ
Definition des Typs aller Array-Elemente

Die typische Form der Array-Verwendung ist die Bearbeitung der Array-Elemente im Rahmen einer oder mehrerer Schleifen.

Beispiel:

```
Dim Arr() as String

Redim Arr(0 To 3)
ArrA(0) = "Tschüß"
ArrA(1) = "Ade"
ArrA(2) = "Auf Wiedersehen"
ArrA(3) = "Hallo"
```

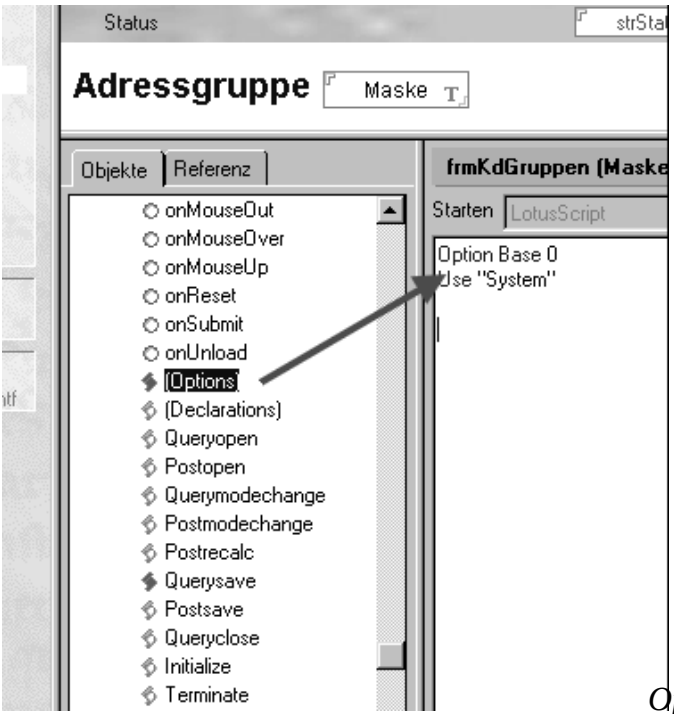
6.3.2.5.1 Option Base

Ob der Index eines Arrays bei 0 oder 1 beginnt, kann unterschiedlich festgelegt werden. Die lesbarste Form ist die direkte Angabe der Grenzen:

```
Dim ArrB(1 To 9) as Integer
```

Sind die Grenzen nicht in der Definition angegeben, hängt der Beginn des Indexwertes von der Definition von „Option Base“ ab. Sie kann im Scriptmodul unter Options hinterlegt werden. Dabei bedeutet:

- Option Base 0
  Der Index beginnt bei 0.
- Option Base 1
  Der Index beginnt bei 1.
- Keine Angabe
  Der Index beginnt bei 0.



Option Base

Beispiele:

```
Dim ArrA(4) as String

' Das Array beginnt mit Element 0 und endet mit '
' Element 3

Option Base 0

Dim ArrB(4) as Integer

' Das Array beginnt mit Element 0 und endet mit '
' Element 3

Option Base 1

Dim ArrC(4) as Currency

' Das Array beginnt mit Element 1 und endet mit '
' Element 4
```

6.3.2.5.2 Das dynamische Array

Ist ein Array ohne Angabe der Elementanzahl oder der Grenzen für die Elemente definiert, so steht es als dynamisches Array zur Verfügung. In dieser Form ist das Array noch nicht verwendbar. Erst wenn Elemente definiert und entsprechende Speicherplätze reserviert wurden, können Daten in dem Array abgelegt werden. Um die gewünschte Anzahl an Elementen zu erhalten, wird sie mit Hilfe der ReDim-Anweisung definiert. Da solche Definitionen an beliebiger Stelle im Programmtext erscheinen können, kann das Array immer wieder dem jeweiligen Bedarf angepasst werden.

Die ReDim-Anweisung hat die Syntax:

ReDim [Preserve] Name (Grenzen) [As Typ]

Dabei bedeuten:

ReDim
Schlüsselwort für die Definition der Elemente

Preserve
Schlüsselwort für den Erhalt der bestehenden Elemente

Name
Name des Arrays

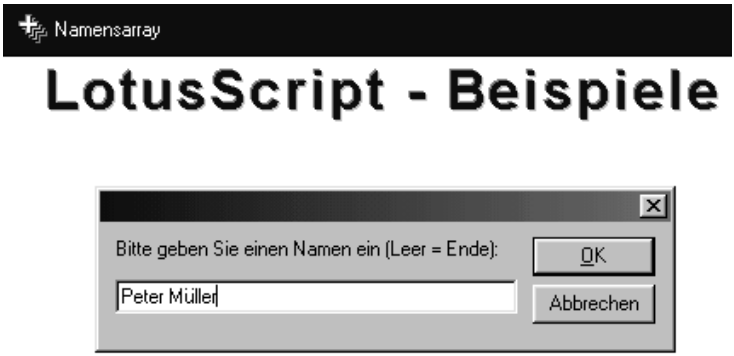
Grenzen
Anzahl der Elemente je Dimension oder Von-/Bis-Werte je Dimension oder keine Angabe (dynamisches Array)

As Typ
Definition des Typs aller Array-Elemente. Der Datentyp der Elemente muss dem Typ der Dim-Anweisung entsprechen, mit der das Array angelegt wurde und kann entfallen.

Beispiele:

```
Dim ArrA() As String
Dim strName As String
Dim Zaehler As Integer

Zaehler = 0

Do
    ' Name abfragen
    strName = _
    Inputbox$("Name (Leer = Ende):")

    ' Wenn Eingabe erfolgt, verarbeiten
    If strName <> "" Then
          ' Array erweitern:
          ' Element 1 = Index 0 usw.
          Redim Preserve ArrA(Zaehler + 1)

          ' Name speichern
          ArrA(Zaehler) = strName

          Index erhöhen
          Zaehler = Zaehler + 1
    End If


' Schleife beenden, wenn keine Eingabe
' erfolgt
Loop Until strName = ""
```

Die Ausführung des Script-Beispiels sieht dann wie folgt aus:



Namenseingabe für die Erweiterung des Arrays

Damit Werte, die in Array-Elementen gespeichert sind, nicht durch die ReDim-Anweisung verloren gehen, muss unbedingt darauf geachtet werden, dass das Schlüsselwort „Preserve“ angegeben wird.

6.3.2.5.3 Funktionen zum Array

Man findet in LotusScript Funktionen zur Bearbeitung von Array-Elementen und solche zur Veränderung des ganzen Arrays:

- ArrayAppend
  Zwei Arrays werden aneinander gehängt.
- ArrayGetIndex
  sucht den Index eines Wertes in einem Array.

- ArrayReplace
  Mit ArrayReplace werden in einem Array diejenigen Werte durch Werte eines zweiten Arrays ersetzt, die denen eines dritten entsprechen. Es werden also alle Werte zweier Arrays miteinander verglichen. Sind sie gleich, so wird der Wert eines dritten Arrays mit dem gleichen Index in das erste Array übernommen.

- Erase
  löscht den Inhalt aller Elemente eines statischen Arrays bzw. entfernt alle Elemente eines dynamischen Arrays. Der Datentyp bleibt erhalten.

- ForAll
  erlaubt die Verarbeitung aller Elemente eines Arrays in Form einer Schleife.

- IsArray
  prüft, ob eine Variable ein Array ist.

- LBound
  liefert die Untergrenze eines Arrays zurück.

- UBound
  liefert die Obergrenze eines Arrays zurück.

6.3.2.6 Der Einsatz von Listen

Listen sind eine allgemeinere Form von Arrays. Sie unterscheiden sich vom Array dadurch, dass sie ihre Elemente nicht über Indices, sondern über Positionsnamen (so genannte ListTags) verwalten. Die Anzahl der Listenelemente wird nicht bei der Definition festgelegt, sondern hängt vom Anlegen und Löschen der einzelnen Elemente ab. Wie das Array kann eine Liste jeweils nur einen Datentyp speichern. Mehr als eine Dimension ist bei Listen nicht verfügbar. Die Syntax der Liste lautet:

Dim [Public | Private | Static] Name List [As Typ]

Dabei bedeuten:

Dim
Schlüsselwort für die Listen-Definition

Public, Private
Gültigkeitsbereich (innerhalb einer Funktion / Prozedur, auf Modulebene usw.)

Static
Obwohl die Definition innerhalb einer Funktion bzw. Prozedur festgelegt ist, behält die Liste ihre Werte, wenn die Funktion / Prozedur verlassen wird.

Name
Name der Liste

List
Konstantes Schlüsselwort zur Definition als Liste

As Typ
Definition des Typs aller Listen-Elemente

Der ListTag ist eine Zeichenkette, gleichgültig ob Ziffern oder Buchstaben verwendet werden. Beim Suchen nach Elementen der Liste wird die Suchreihenfolge verwendet, die unter „Option Compare“ festgelegt ist. Wird die Option nicht gesetzt, verwendet LotusScript die Groß-/Kleinschreibung bei der Suche in Listen.

Beispiele:

```
Option Compare Case

Dim MeineListe List as Integer
Dim DeineListe List as String
Dim strTier as String
Dim intZahl as Integer

MeineListe("1") = 234
MeineListe("2") = 28

DeineListe("Elefanten") = "Großer Elefant"
DeineListe("Mäuse") = "Kleine Maus"

strTier = DeineListe("Elefanten")
intZahl = MeineListe("2")
```

6.3.2.6.1 Funktionen für Listen

Für die Verarbeitung von Listen stehen nur wenige spezielle Funktionen zur Verfügung:

- Erase
  löscht ein Element einer Liste oder eine ganze Liste, je nachdem was als Parameter angegeben wurde.
- ForAll
  erlaubt die Verarbeitung aller Elemente einer Liste in Form einer Schleife.

- IsElement
  liefert TRUE zurück, wenn der angegebene String ein ListTag der genannten Liste ist, oder FALSE im anderen Fall.

- IsList
  prüft, ob die angegebene Variable eine Liste ist.

- ListTag
  gibt den Namen des gerade bearbeiteten Listenelementes in einer ForAll-Schleife zurück.

- TypeName
  liefert den Datentyp der angegebenen Liste.

Beispiele:

```
' Liste definieren
Dim Feld List as String
Dim strTyp as String

' Werte zuweisen
Feld("Name") = "Vorname"
Feld("Label") = "Vorname"
Feld("Laenge") = "15"
Feld("Pruefung") = "*NO"

' Einen Eintrag entfernen
Erase Feld("Pruefung")

' Prüfen, ob der Eintrag fehlt und ihn ergänzen
if NOT IsElement(Feld("Pruefung")) then
   Feld("Pruefung") = "*NO"
else
   Feld("Pruefung") = "*YES"
end if


' Alle Elemente verarbeiten
ForAll x In Feld
   Print ListTag(x)
End ForAll

' Typ der Liste ermitteln
strTyp = TypeName(Feld)

' Liste löschen
Erase Feld
```

6.3.2.7 Eigene Typen definieren

Wer mit den Datentypen von LotusScript nicht auskommt, der kann sich eigene Typen definieren. Die entsprechende Anweisung heißt „Type“. Die Syntax lautet:

```
[Public | Private] Type Name
      Elementvariable des Typs
End Type
```

Dabei bedeutet:

Public, Private
Gültigkeitsbereich (innerhalb einer Funktion / Prozedur, auf Modulebene usw.).

Type
Schlüsselwort für die Kennzeichnung als Type

Name
Name des Typs

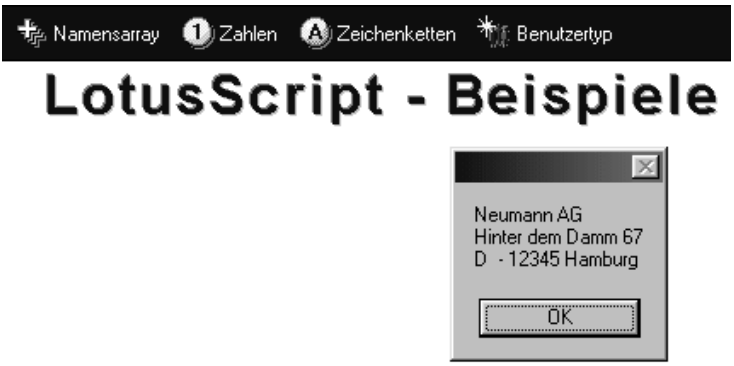
Elementvariable des Typs
Beliebige Variable, statische Arrays oder Variable mit benutzerdefinierten Datentypen

Benutzerdefinierte Datentypen können mehr als einen Standard-Datentyp aufnehmen. Sie sind deshalb besonders interessant, wenn tabellenartige Daten verarbeitet werden sollen, bei denen die Spalten unterschiedliche Datenarten enthalten (Dateiverarbeitung und Druckoperationen). Der Zugriff auf die Elementvariablen eines benutzerdefinierten Typs erfolgt unter Verwendung der „Punkt-

Syntax“. Es wird zunächst der Name der Variable und dann durch Punkt getrennt der Name der Elementvariable verwendet.

Beispiele:

```
' Typ definieren
Type Adresse
   Vorname As String *15
   Name As String *20
   Strasse As String
   Land As String *3
   Plz As Long
   Ort As String
End Type

' Variable mit dem neuen Typ definieren
Dim Firmen As Adresse
Dim strMsg As String

' Elementvariable füllen
Firmen.Vorname = ""
Firmen.Name = "Neumann AG"
Firmen.Strasse = "Hinter dem Damm 67"
Firmen.Land = "D"
Firmen.Plz = 12345
Firmen.Ort = "Hamburg"


' Meldung aus Elementvariablen
' zusammensetzen
strMsg = _
   Firmen.Name & Chr$(10) & _
   Firmen.Strasse & Chr$(10) & _
   Firmen.Land & "-" & _
   Str$(Firmen.Plz) & " " & _
   Firmen.Ort

' Meldung anzeigen
Msgbox strMsg
```



Beispiel für einen benutzerdefinierten Datentyp

6.3.2.8 Programme steuern

Wie wäre das Leben schön, wenn alles einfach der Reihe nach abgearbeitet werden könnte. Aber leider gibt es zu jeder Regel mindestens eine Ausnahme und manches muss mehrfach wiederholt werden. LotusScript wird diesen Anforderungen gerecht und beinhaltet alle Anweisungsarten, die man für die Steuerung von Programmen benötigt. Zum einen sind das Anweisungen zur Behandlung von Codes, die nur unter einer oder mehreren Bedingungen ausgeführt werden sollen. Zum anderen sind es Schleifenanweisungen, die in einer festen oder einer bedingten Anzahl von Durchläufen abgearbeitet werden sollen.

Die Anweisungen zur bedingten Verzweigung heißen:

If Then ...

Select Case ...

Zur Steuerung von Schleifen kennt LotusScript:

Do
While
For
Forall

Dazu kommen noch Sprungbefehle:

Goto
Gosub
If ... Goto
On ... Goto
On Gosub

Zur vorzeitigen Beendigung von Schleifen stehen ebenfalls Anweisungen zur Verfügung:

Exit
End

Die Steuerungsanweisungen können wiederum ineinander verschachtelt genutzt werden.

6.3.2.8.1 Bedingte Programmsteuerung

Sowohl die If- als auch die Select-Anweisung erlauben die Steuerung von bedingten Programmanweisungen. Die If-Anweisung wird in der Regel als „entweder/oder"-Verarbeitung eingesetzt. Die Select-Anweisung dient der Abarbeitung von Bedingungen mit mehreren Alternativen.

6.3.2.8.1.1 Die If-Anweisung

Die If-Anweisung gruppiert Anweisungen, die unter bestimmten Bedingungen ausgeführt werden. Die Syntax der If-Anweisung lautet:

If *Bedingung erfüllt* **Then**
    Anweisungen
Elseif *Bedingung erfüllt* **Then**
    Anweisungen
Elseif *Bedingung erfüllt* **Then**
    Anweisungen
Else
    Anweisungen
End If

Die einfachste Form der If-Anweisung umschließt einen Block von Anweisungen, die nur unter der genannten Bedingung ausgeführt werden. Ist die Bedingung erfüllt, so liefert ihre Formel den Wert TRUE und die Anweisungen bis zum „End If" werden ausgeführt. Ist die Bedingung nicht erfüllt, wird die erste Anweisung nach „End If" ausgeführt.

Ist nur eine Anweisung auszuführen, kann sie in der gleichen Zeile und ohne Verwendung von „End If“ geschrieben werden.

Programmtext

```
if zahl = 0 then print „Null“

if zahl > 45 then
    erg = zahl + 17
    v = zahl - erg
end if
```

Die erweiterte Form der If-Anweisung verwendet „Else“, um die Anweisungen zur Ausführung zu bringen, die die If-Bedingung nicht erfüllen (also FALSE ergeben).

Programmtext

```
If strSelect = wALL Then
    strView = wVW_ADRESS_WAHL
Else
    strView = wVW_COMPANY_WAHL
End If
```

In diesem Beispiel wird einer Variable in Abhängigkeit von dem Inhalt einer anderen Variablen ein unterschiedlicher Wert zugewiesen.

Die nächste Stufe der Erweiterung liegt darin, dass eine zusätzliche Bedingung formuliert wird, die erst dann geprüft wird, wenn die ursprüngliche If-Bedingung nicht erfüllt ist. Die neue Bedingung muss dabei in keinem Zusammenhang zur If-Bedingung stehen.

Programmtext

```
if Parm = „Kunde“ then
    strName = docAdr.Name(0)
elseif DEFAULT = TRUE then
    strName = “Testkunde”
else
    strName = “”
end if
```

Die Else-Bedingung dient dann dazu, alle Fälle abzudecken, für die keine der vorherigen Bedingungen zugetroffen hat.

6.3.2.8.1.2 Die Select-Anweisung

Für den Fall, dass für unterschiedliche Inhalte einer Variablen auch unterschiedliche Anweisungen abgearbeitet werden sollen, verwendet LotusScript die Select-Anweisung. Dabei wird mit „Select“ eine Variable ausgewählt und über „Case“ die Gruppe der Anweisungen, die für einen bestimmten Wert der Variablen gültig sein soll:

Select Case *Ausdruck*
    Case *Werte*
        Anweisungen
    Case Werte
        Anweisungen

    Case Else
        Anweisungen
End Select

Die Select-Anweisung berechnet den Ausdruck und vergleicht das Ergebnis mit dem Inhalt der Werte der Case-Zeilen. Sobald ein Wert gefunden wird, werden die dazugehörigen Anweisungen ausgeführt. Alle weiteren Case-Zeilen werden nicht mehr verglichen. Die Ausführung wird nach der “End Select”-Anweisung fortgesetzt.

Programmtext

```
if left$(doc.DbPfad(0), 1) = „*“ then

    Select Case docCur.DbPfad(0)
      Case “*DATA”
            strDbPfad = “”
      Case “*START”
            strDbPfad = strStartPfad
      Case “*ORG”
            strDbPfad = strFirmaPfad
      Case “*GROUP”
            strDbPfad = strGroupPfad
      Case Else
            strDbPfad = “”
    End Select

else
    strDbPfad = trim(doc.DbPfad(0))

    if Right$(strDbPfad, 1) <> “\” then
      strDbPfad = strDbPfad & “\”
    end if

end if
```

In diesem Beispiel werden verschachtelte If- und Select-Anweisungen dazu genutzt, den Wert einer Variablen festzulegen. Je nachdem, ob in dem Feld DBPfad des aktuellen Dokumentes ein Standardbegriff (beginnt mit "*") oder ein fester Name angegeben ist, wird weiter ausgewertet bzw. der Inhalt des Feldes auf syntaktische Richtigkeit geprüft. Handelt es sich um einen Standardbegriff, so wird dieser in der Select-Anweisung durch seine Entsprechung ersetzt.

Der Ausdruck einer Select-Anweisung kann ein Wert oder eine Formel sein. Bei den Case-Zeilen können ein oder mehrere Werte bzw. ein Wertebereich angegeben werden. Handelt es um einen boolschen Wert, so wird das Schlüsselwort „Is" verwendet.

Programmtext

```
select case Variable
    case 0 to 3
      print "Wert zu klein"
    case 4,7,9
      print „Wert ist ungültig"
    case is >= 1000
      print „Wert zu groß"
    case else
      print „Gültiger Wert"
end select
```

6.3.2.8.2 Programmschleifen

Bei den Programmschleifen kann man zwischen den bedingten Schleifen und den Zählschleifen unterscheiden. Beide Schleifenarten sollen dazu dienen, eine Anzahl von Anweisungen mehrfach zu durchlaufen und dann die Verarbeitung nach dem Ende der Schleife fortzusetzen. Typischer Vertreter für die bedingten Schleifen ist die While- und für Zählschleifen die For-Schleife.

6.3.2.8.2.1 Die While-Schleife

Das Merkmal der While-Schleife ist die Ausführungsbedingung am Beginn (dem Kopf) der Schleife. Das bedeutet, dass – während der Ausführung der innerhalb der Schleife angesiedelten Anweisungen – die Kopfbedingung ungültig werden muss. Ist die Bedingung immer erfüllt, wird die Schleife endlos durchlaufen. Die genaue Syntax der While-Schleife lautet:

While *Bedingung erfüllt*
    Anweisungen
Wend

Die While-Schleife entspricht der Do-While-Schleife (siehe Do-Schleife).

Programmtext

```
Dim i As Integer

‘// Array mit 7 Elementen (0 bis 6)
Dim Titel(7) As String
Dim msg As String

i = 0
msg = “”

While i < 7
    ‘// Array füllen
    Titel(i) = „Titel „ & i & „ „

    ‚// String Bilden
    msg = msg & Titel(i)

    ‚// Zähler erhöhen
    i = i + 1
Wend

Msgbox msg
```

Das Beispiel füllt ein Array mit dem Text „Titel „ und dem Zähler. In der Anweisung „Titel(i) = „Titel „ & i“ wird ein String erstellt und in das jeweilige Array-Element gestellt. Außerdem wird ein String mit den Elementen zusammengesetzt. Die Variable i ist dabei vom Typ Integer und wird durch die Verwendung von „&“ in einen String eingebunden.

Erstellen eines Strings mit der While-Schleife

Die Bedingung muss nicht mit einem Zählvorgang verbunden werden. Die Schleife könnte z. B. auch dadurch beendet werden, dass eine Eingabe durch den Benutzer erfolgt.

6.3.2.8.2.2 Die Do-Schleife

Die Do-Schleife ist wesentlich flexibler als die While-Schleife. Sie kann sowohl durch eine Kopf- als auch durch eine Fußbedingung gesteuert werden. Die reine Do-Schleife ohne Bedingung ist eine Endlos-Schleife, die nur durch spezielle Befehle verlassen werden kann. Die Do-Schleife hat die Syntax:

Do [**While | Until** *Bedingung erfüllt*]
Anweisungen
Loop [**While | Until** *Bedingung erfüllt*]

Bei der kopfgesteuerten Do-Schleife kann entweder über „While" geprüft werden, ob die Bedingung erfüllt bleibt, oder ob sie schon erfüllt ist. In beiden Fällen wird die Schleife nicht durchlaufen, wenn die Bedingung nie erfüllt ist.

Bei der fußgesteuerten Schleife werden die Anweisungen innerhalb der Schleife mindestens einmal durchlaufen.

Programmtext

```
Set doc = col.GetFirstDocument

'// Kopfgesteuerte Schleife:
,// solange ein Dokument gefunden wird,
,// wird es mit aktuellen Werten gefüllt
do While Not doc Is Nothing

    With doc
    .AdrType = uidocCur.FieldGetText("Typ")
```

```
    .AdrNumber = uidocCur.FieldGetText("ANr")
    .AktStatus = uidocCur.FieldGetText("Status")
    .save True, False
    End With

    i = i + 1
    Set doc = col.GetNextDocument(doc)

loop

'// Endlos-Schleife, die durch eine spezielle
,// Anweisung - exit - beendet wird
do
    intPos = instr(intStart, strPath, "\")

    if intPos = 0 then exit do

    intBackSlash = intPos
    intStart = intPos + 1
loop
```

Das erste Beispiel zeigt die Verarbeitung einer Dokumentensammlung, die eines nach dem anderen geändert werden. Sobald alle Dokumente abgearbeitet sind, wird die Schleife verlassen.

6.3.2.8.2.3 Die For-Schleife

Die For-Schleife ist in LotusScript sehr einfach ausgeführt, wenn man sie z. B. mit C oder Java vergleicht. Es wird angegeben, von welchem Startwert aus und in welchen Schritten eine Zählvariable

erhöht oder vermindert wird sowie bei welchem Wert der Zählvariablen die Schleife verlassen werden soll.

For *Zählvariable* **=** *Startwert* **To** *Endwert* [**Step** *Intervall*]
    Anweisungen
Next [*Zählvariable*]

Der Startwert wird bei jedem Durchlauf um den Wert von Step erhöht, und zwar solange, bis der Endwert erreicht ist. Wird Step nicht angegeben, so wird 1 als Erhöhungswert verwendet. Soll der Endwert durch Subtraktion ermittelt werden, so ist Step mit negativem Vorzeichen anzugeben.

Programmtext

```
Dim i as integer
Dim j as integer

for i = 1 to 10

    for j = 3 to 9 step 3

      print "i hat den Wert "; i; _
            „ j hat den Wert „; j
    next j

next i
```

In dem Beispiel wird in einer äußeren Schleife der Zähler i jeweils um 1 erhöht und damit 10-mal durchlaufen. In der inneren Schleife wird der Zähler j um 3 erhöht und deshalb nur 3-mal durchlaufen. Die Ausgabe des Print-Befehls erfolgt in der Statuszeile.

6.3.2.8.2.4 Die Forall-Schleife

Die Forall-Anweisung kann nur im Zusammenhang mit bestimmten Sprachelementen eingesetzt werden. Das ist dadurch verursacht, dass die Forall-Schleife keine eigene Zählvariable verwendet, sondern die Anzahl der Durchläufe von der Elementzahl einer Array oder einer Liste bestimmt wird. Für jedes Element der Array oder Liste werden die Anweisungen innerhalb der Schleife ausgeführt. Ihre Syntax lautet:

Forall *Zählervariable* **In** *Array oder Liste*
Anweisungen
End Forall

Eine weitere Besonderheit der Forall-Schleife besteht darin, dass die Zählvariable nicht mit „Dim Zählvariable“ definiert wird, sondern dass ihre Definition automatisch erfolgt. Sie dient ausschließlich der Verwendung innerhalb der Schleifenanweisungen als Ersatz für das jeweils verwendete Element.

Programmtext

```
Dim Obst(5) as string

Obst(0) = “Apfel”)
Obst(1) = «Birne»)
Obst(2) = «Pfirsich»)
Obst(3) = „Pflaume“)
Obst(4) = „Kirsche“)

forall i in Obst
    print i
end forall
```

6.3.2.8.3 Die Goto-Anweisung

LotusScript bietet die Möglichkeit, mit Hilfe von Goto-Anweisungen zu Sprungmarken (Label) innerhalb eines Programmes zu verzweigen. Es hat sich wohl inzwischen herumgesprochen, dass die intensive Verwendung solcher Befehle nicht nur schlechter Programmierstil, sondern auch äußerst verwirrend ist. Es gibt trotzdem Situationen, in denen man Goto-Anweisungen verwendet – z. B. bei der Behandlung von Fehlern.

Die Varianten der Goto-Anweisung von Lotus Script sind:

Goto *Sprungmarke*
: Unbedingte Verzweigung zu einer Sprungmarke.

If *Bedingung erfüllt* **Goto** *Sprungmarke* **Else ... End If**
: Verzweigung zu einer Sprungmarke, wenn eine Bedingung erfüllt ist.

On *Ereignis* **Goto** *Sprungmarke*
: Verzweigung bei Eintreten eines Ereignisses zu einer Sprungmarke.

Gosub ... Return
: Verzweigung zu einer Anweisungsgruppe innerhalb einer Prozedur.

On *Ausdruck* **Gosub**
: Verzweigung zu einer Anweisungsgruppe bei Eintreten eines Ereignisses.

Return
: Rücksprung nach einem Goto bzw. GoSub zu der nächsten Anweisung nach der Verzweigung.

Die Sprungmarke als Ziel der Verzweigung muss am Anfang einer Zeile stehen und mit einem Doppelpunkt abgeschlossen werden.

Programmtext

```
    ,// Fehlerbehandlung über on Error
    on Error goto Fehler

    strS = wGetStartPfad & wDB_START
    set dbCur = sesCur.CurrentDatabase
    strServ = dbCur.Server
    set dbS = new NotesDatabase(strServ, strS)
    set vwUser = dbStart.GetView(wVW_USERPROFIL)
    set docUser = vwUser.GetFirstDocument

    wGetStart = strStartDb

    ,// Funktion verlassen, wenn kein Fehler
    ,// vorliegt
    exit function

,// Sprungmarke
Fehler:
    ,// Fehlerroutine aufrufen
```

Eine wichtige Funktion der Goto-Anweisung besteht darin, als Fehlerbehandlungswerkzeug zu arbeiten. Mit Hilfe der Anweisung „On Error Goto ...“ wird zu einer Fehlerbehandlungsroutine verzweigt. Je nach Art des Fehlers kann die Arbeit weitergehen oder die Funktion bzw. Prozedur verlassen werden.

6.3.2.8.4 Spezielle Verzweigungsanweisungen

In einigen Fällen sollen Ausnahmen von der normalen Verarbeitung möglich sein. Eine Schleife soll abgebrochen oder eine Prozedur verlassen werden. Für solche Fälle enthält LotusScript die Anweisungen:

End
Exit

Die End-Anweisung führt dazu, dass die aktuelle Prozedur und alle, durch die sie aufgerufen wurde, beendet werden. Ihre Syntax lautet:

End [*Rückgabewert*]

Die Exit-Anweisung beendet eine Schleife mit Do, For, Forall sowie eine Prozedur oder Funktion. Ihre Syntax lautet:

Exit [For | Do | Forall | Sub | Function | Property]

Programmtext

```
Dim wspCur As New NotesUiWorkspace
Dim uidocCur As NotesUiDocument

On Error Resume Next
Set uidocCur = wspCur.CurrentDocument

‘// Wenn das Dokument sich nicht im
,// Bearbeitungsmodus befindet, wird die
,// Prozedur verlassen
If Not uidocCur.EditMode Then Exit Sub
```

Ob man solche Anweisungen verwendet, bleibt dem Einzelnen überlassen. Zur Vermeidung von Exit-Anweisungen können z. B. If-Anweisungen dazu verwendet werden, den Anweisungsteil, der nicht ausgeführt werden soll, zu umgehen.

6.3.2.9 Benutzerkommunikation

Programme, die nicht von einem Server, sondern interaktiv von einem Benutzer ausgeführt werden, können mit Hilfe einiger LotusScript-Anweisungen mit diesem Benutzer Kontakt aufnehmen. Sie können Informationen ausgeben und auch die Eingabe von Daten anfordern. Die Anweisungen dazu sind z. T. relativ komplex und erlauben es, dem Programmierer – mit wenigen Angaben – Benutzerangaben abzufragen. Der Domino-Designer kennt folgende Anweisungen zur Benutzerkommunikation:

- **Beep**
  Akustisches Signal am lokalen Rechner, das von dessen Lautsprecher abhängt.
- **Dialogbox**
  Anzeige eines maskenartigen Dialoges, dessen Aussehen frei gestaltbar ist.
- **Inputbox**
  Standarddialog zur Abfrage eines Wertes am Bildschirm.
- **MsgBox oder MessageBox**
  Ausgabe einer Meldung am Bildschirm
- **Picklist**
  Auswahlliste von Dokumenten aus einer Ansicht.
- **Print**
  Ausgabe einer Meldung in der Statuszeile
- **Prompt**
  Liste von Werten zur Auswahl

Während die Anweisungen Inputbox, Messagebox und Print meist einfach zu verwenden sind, erfordern Dialogbox, Picklist und Prompt einiges an Vorarbeit, damit sie ihre Wirkung zeigen. So muss z. B. für die Dialogbox-Anweisung zunächst eine Maske oder Teilmaske erstellt werden, die das Erscheinungsbild des Dialoges definiert. Bei Dialogbox, Picklist und Prompt handelt es sich außerdem nicht um eigenständige Funktionen, sondern um Methoden der Notes-Klasse „NotesUiWorkspace“. An dieser Stelle sollen nur die Anweisungen der LotusScript-Sprache beschrieben werden.

6.3.2.9.1 Der Print-Befehl

Der Print-Befehl erlaubt die einfache Ausgabe von Informationen in der Statuszeile des aktuellen Fensters. Die Syntax des Befehls ist:

Print *Ausgabetext oder Ausdruck*

Der Ausgabetext kann Variable, Konstante, einen Ausdruck oder gar nichts enthalten. Die einzelnen Werte können durch Komma, Leerzeichen sowie Semikolon getrennt werden. Die Wirkung der Trennzeichen ist:

Komma
Zwischen die Werte wird ein Tabulatorsprung eingefügt.

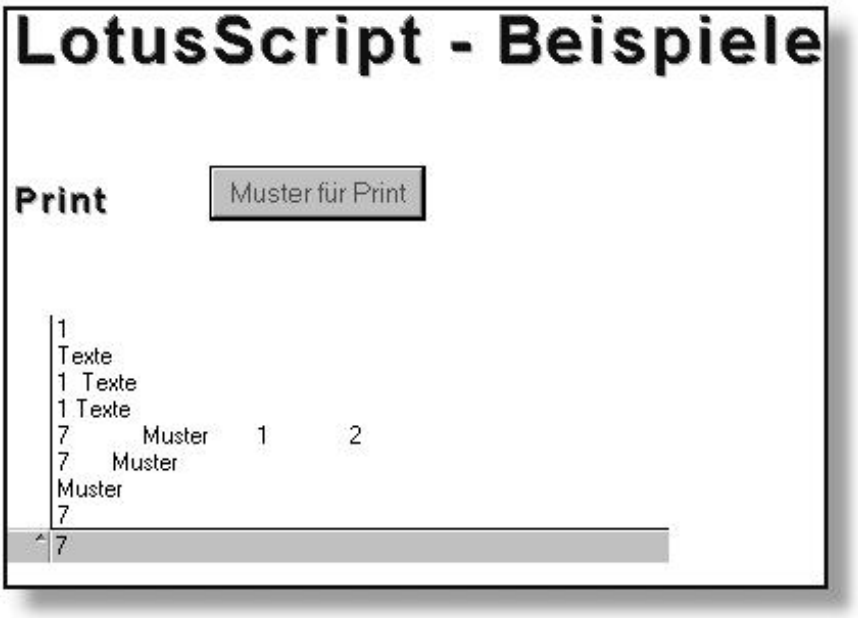
Leerzeichen und Semikolon
Die Werte werden ohne Abstand aneinandergesetzt.

Ähnliche Effekte lassen sich mit den Funktionen Spc(x) und Tab(x) erreichen. Variable werden gemäß ihrem Inhalt dargestellt. Leere Variable werden ausgelassen bzw. als Leerzeile ausgegeben.

Ist der Debugger eingeschaltet, wird die Print-Anweisung in dem Ausgabefenster ausgeführt. Bei der Ausführung auf einem Server wird das Ergebnis in die Datei „Notes.log“ umgeleitet und auf der Konsole angezeigt.

Programmtext

```
Dim i As Integer
Dim a As String

i = 7
a = « Muster»

Print 1
Print «Texte»
Print 1; « Texte»
Print 1 & « Texte»
Print i, a, 1, 2
Print i Spc(5) a
Print a Tab(4) i
```



Print-Ausgabe in der Statusleiste

6.3.2.9.2 Die Inputbox

Die Inputbox-Anweisung erlaubt auf einfache Weise die Abfrage einer Benutzereingabe am Bildschirm. Es muss keine Maske entworfen werden. Inputbox verwendet seine Standardelemente. Die Anweisung hat folgendes Aussehen:

> Ergebnis = **Inputbox[$]** (Text [, [Titel] [, [Vorgabewert] [, x-Position, y-Position]]])

Dabei bedeutet:

- **Ergebnis**
  Intputbox liefert als Rückgabewert eine Variable vom Typ Variant, dagegen hat sie als Ergebnis eine Variable vom Typ String. Bricht der Benutzer die Eingabe ab, so wird ein leerer String zurückgeliefert.

- **Text**
  Der Text dient zur Angabe einer Eingabeaufforderung (z. B. „Bitte geben Sie Ihren Namen ein:“).

- **Titel**
  Der Titel wird im Dialogfenster als Überschrift angezeigt.

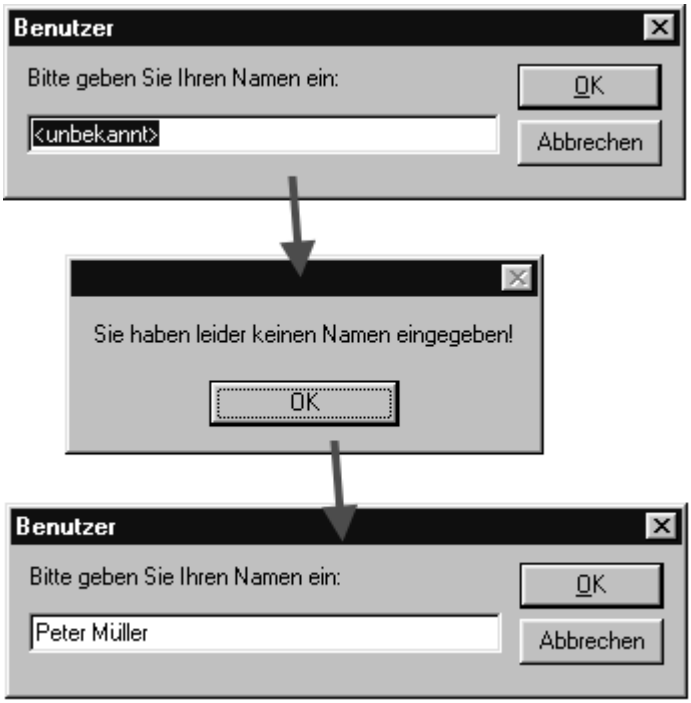
- **Vorgabewert**
  Um dem Benutzer die Arbeit zu erleichtern, kann im Vorgabewert angegeben werden, was im Erfassungsfeld steht, wenn sich das Dialogfenster öffnet.

- **x-Position / y-Position**
  Über die Positionsangabe in Pixel kann eingestellt werden, an welcher Stelle der Dialog auf dem Bildschirmfenster stehen soll.

Der Inputbox-Dialog kann durch Anklicken der Schaltfläche „OK“ oder „Abbrechen“ beendet werden. Danach kann das Ergebnis ausgewertet werden.

Programmtext

```
Const MSG = _
    „Sie haben leider keinen Namen eingegeben!“
Dim Ergebnis As String
Dim Text As String
Dim Vorgabe As String
Dim xPos As Integer, yPos As Integer

Text = „Bitte geben Sie Ihren Namen ein: „
Titel = „Benutzer“
Vorgabe = „<unbekannt>“
xPos = 500
yPos = 500

Do
    Ergebnis = Inputbox$(Text, Titel, _
      Vorgabe, xPos, yPos)
    If Trim(Ergebnis) = „<unbekannt>“ Then
      Msgbox MSG
    Elseif Trim(Ergebnis) = “” Then
      Exit Do
    End If

Loop While Trim(Ergebnis) = “<unbekannt>”
```



Inputbox

Das Beispiel zeigt die einzelnen Bestandteile der Inputbox. Als Vorgabewert wird „<unbekannt>“ an den Dialog übergeben. Klickt der Benutzer „OK“ an, ohne eine echte Eingabe zu machen, erscheint eine Aufforderung, die Eingabe zu berichtigen. Damit kehrt das Programm zur Inputbox zurück. Klickt der Benutzer auf „Abbrechen“, so erhält die Variable „Ergebnis“ den Wert „“ und die Schleife wird verlassen. Wenn der Benutzer ei-

nen anderen Wert eingibt, wird die Schleife ebenfalls verlassen und „Ergebnis“ steht jetzt zur Verarbeitung zur Verfügung.

6.3.2.9.3 Die Messagebox

Die Messagebox verfügt über einige Parameter mehr als die Inputbox und erlaubt entsprechend mehr Variationen. Im Gegensatz zur Inputbox lässt die Messagebox aber keine Eingabe eines Wertes durch den Benutzer zu. Die Messagebox hat die Syntax:

Messagebox Nachricht [, [Schaltfläche + Symbol + Vorgabeschaltfläche + Modus] [, Titel]]

Dabei bedeutet:

- **Nachricht**
  Text, der als Nachricht angezeigt werden soll.
- **Schaltfläche**
  Zahl, die folgende Werte annehmen kann:

0	OK
1	OK und Abbrechen
2	Beenden, Wiederholen und Ignorieren
3	Ja, Nein und Abbrechen
4	Ja und Nein
5	Wiederholen und Abbrechen

- **Symbol**
  Zahl, die folgende Werte annehmen kann:

16	Stop-Symbol
32	Fragezeichen

48 Ausrufezeichen
64 Symbol für Information

- **Vorgabeschaltfläche**
  Zahl, die folgende Werte annehmen kann:

  0 erste Schaltfläche
  256 zweite Schaltfläche
  512 dritte Schaltfläche

- **Modus**
  Zahl, die folgende Werte annehmen kann:

  0 Die aktuelle Anwendung wird unterbrochen, bis der Benutzer eine Schaltfläche angeklickt hat.
  4096 Alle Anwendungen werden unterbrochen, bis der Benutzer eine Schaltfläche angeklickt hat (modal).

  Die Werte von Schaltfläche, Symbol, Vorgabeschaltfläche und Modus werden addiert und als eine Zahl eingetragen. Anstelle der Zahlenwerte können auch die Standard-Konstanten-Namen von LotusScript verwendet werden (z. B. MB_OK anstelle von 0 bei der Schaltfläche), die in der Datei „LSCONST.LSS“ enthalten sind.

- **Titel**
  Text, der als Überschrift des Dialogfensters erscheint.

Die Messagebox-Anweisung ist in zwei Formen vorhanden: als Befehl, so wie sie oben beschrieben wurde, oder als Funktion. Die Funktion Messagebox liefert ein Ergebnis in Form einer Integer-Variablen zurück. Außerdem werden bei der Funktion die Parameter in Klammern gesetzt.

Das Ergebnis hat folgende Bedeutung:

1	OK
2	Abbruch
3	Beenden
4	Wiederholen
5	Ignorieren
6	Ja
7	Nein

Das Programm kann das Ergebnis weiterverarbeiten und damit auf die Benutzereingabe reagieren.

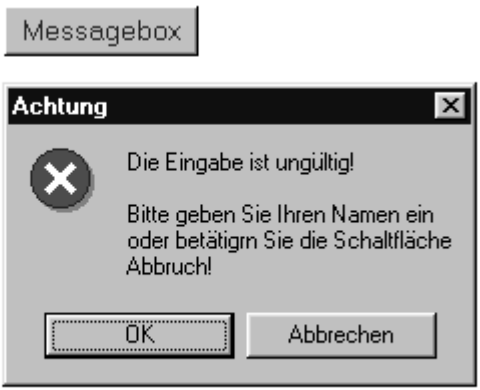
Der Nachrichtentext wird in dem Dialogfenster automatisch umbrochen. Soll ein bestimmter Zeileninhalt erreicht werden, so kann an der Position des gewünschten Zeilenumbruchs ein Zeilenwechsel (das Zeichen „chr(10)“) eingefügt werden.

Alternativ zur „Messagebox“ kann auch der in anderen Basic-Dialekten übliche Name „Msgbox“ verwendet werden. Das gilt sowohl für den Befehl als auch für die Funktion.

Programmtext

```
If  Trim(Ergebnis) = „<unbekannt>“ Then

msg = „Die Eingabe ist ungültig!“ & Chr(10) _
& Chr(10) & _
„Bitte geben Sie Ihren Namen ein“ & _
Chr(10) & _
„oder betätigen Sie die Schaltfläche“ & _
Chr(10) & „Abbruch!“

Msgbox msg, MBOKCANCEL + MB_ICONSTOP, „Achtung“

Elseif Trim(Ergebnis) = „“ Then
    Exit Do
End If
```

Msgbox



Messagebox

Das Beispiel erweitert die Ausgabe der Messagebox um ein Symbol und die Schaltflächen „OK“ und „Abbrechen“. Der Nachrichtentext trennt die Zeilen mit chr(10).

6.3.3 Fehlerbehandlung und -protokollierung

Hin und wieder soll es vorkommen, dass einem Programmierer ein Fehler unterläuft. Das Schlimme dabei ist, dass solche Fehler nicht ohne Weiteres erkennbar sind. Besonders unangenehm sind Fehler, die unter die Kategorie „Absturz" fallen. Aus unerfindlichen Gründen verweigert das Programm die Weiterarbeit und beendet eine Routine, das ganze Programm oder sogar den Betrieb des Rechners. Um sich vor solchen Vorkommnissen zu schützen, bietet LotusScript dem Programmierer die Möglichkeit, auf Ausnahmesituationen zu reagieren und programmtechnisch Schlimmeres zu verhüten. Tritt ein Ausnahmefehler auf, so kann eine „On Error ..."-Anweisung ausgeführt werden. Diese Anweisung kann über eine Sprungmarke zu einer Gruppe von Anweisungen geführt werden, in der Informationen zu dem Ausnahmefehler abgefragt und bearbeitet werden können. Für die Verarbeitung von solchen Informationen sind in LotusScript folgende Anweisungen enthalten:

- **Erl**
  Gibt die Nummer der Zeile an, bei deren Ausführung der Fehler aufgetreten ist.
- **Err**
  Gibt den Fehlercode des aufgetretenen Fehlers zurück und ermöglicht es, eigene Fehlercodes zu erzeugen.
- **Error**
  Liefert den Fehlertext des aktuellen Fehlers zurück.
- **On Error**
  Verzweigt zu der angegebenen Sprungmarke, wenn ein Ausnahmefehler auftritt.
- **Resume**
  Rücksprung nach der Fehlerbehandlung.

Programmtext

```
‘ // Fehlerbehandlung
NoCollect:
    strMsg = wL_NO_COLLETION_DEF & Chr$(10) & _
    wL_FEHLER & Space(1) & Str$(Err) & _
    Space(1) & wMINUS & Space(1) & Error$ & _
    wL_IN_ZEILE & Erl
    Print strMsg

    ‘ // Prozedur verlassen
    Exit Sub

NoZielDb:
    If Err = wDATABASE_NOT_OPEN Then
      Set dbZ = dbQ.CreateCopy(strServ, _
            strZDb)
      dbZ.Title = _
            docC.TargetDataBaseTitle(0)
      Set vwZ = dbZ.GetView(wVW_ALL_DOCID)
      Set docP = _
            dbQ.GetProfileDocument(wDTYP)
      strTyp = docP.DatenbankTyp(0)
      Set docProfile = _
            dbZ.GetProfileDocument(wDTYP)
      docP.DatenbankTyp = strTyp
      docP.save True, False
      Print wL_PM_DATENBANK_ANGELEGT
    End If
```

```
    ' // Ausführung fortsetzen
    Resume Next

' //
End Sub
```

Eine Fehlerbehandlungsroutine wird mit Resume, Exit Sub, Exit Function oder Exit Property beendet.

Das Domino-System nutzt zur Dokumentierung von ausgeführten Tätigkeiten und aufgetretenen Fehlern spezielle Datenbanken. Eine davon ist die Datenbank „log.nsf". Diese Datenbank kann auch zur Dokumentation von Fehlern im Programmablauf genutzt werden. Daneben gibt es die Möglichkeiten: eine Textdatei zu verwenden, über die Klasse „NotesLog" in die eigene Fehlerdatenbank zu schreiben oder e-Mails mit Fehlernachrichten zu versenden.

6.3.3.1 Anweisungen zur Fehlerbehandlung

Zum Beginn einer Prozedur wird zur Fehlerbearbeitung die Anweisung „On Error ..." eingesetzt. Dadurch wird erreicht, dass zur Laufzeit des Programmes Fehler registriert und verarbeitet werden. Die genaue Syntax der Anweisung „On Error" lautet:

On Error [Fehlercode] {**Goto** Sprungmarke | **Resume Next** | **Goto** 0}

Dabei bedeutet:

- **Fehlercode**
  Nummer des aufgetretenen Fehlers als Integer-Zahl. Die Nummer wird erkannt, solange die „On Error"-Anweisung gültig ist. Wird keine Nummer angegeben, bezieht sich die Anweisung auf alle Fehler, die auftreten.

- **Goto Sprungmarke**
  Bei der Verwendung von „Goto" verzweigt das Programm, sobald ein Fehler auftritt zu der angegebenen Sprungmarke. Dort kann der Fehler ausgewertet und die weitere Bearbeitung geregelt werden. Für den Programmablauf gilt nach der „Goto"-Anweisung, dass der Fehler aufgehoben wurde.

- **Resume Next**
  Mit der „Resume Next"-Anweisung wird das Programm aufgefordert, mit der auf den Fehler folgenden Zeile weiterzumachen. Der Fehler wird dadurch nicht ausgewertet, aber als aufgehoben angesehen.

- **Goto 0**
  „Goto 0" bewirkt, dass die Fehlerbehandlung nicht in der aktuellen Funktion oder Prozedur behandelt wird. Die Verarbeitung geht mit einem Rücksprung in die aufrufende Funktion oder Prozedur weiter.

Während normalerweise das Programm durch einen Fehler sofort beendet wird, kann über die Fehlerbehandlung das Programm auch nach Auftreten eines Fehlers fortgesetzt werden. Wird eine Funktion oder Prozedur verlassen, verliert die „On Error"-Anweisung ihre Wirkung.

Tritt in einem Programm zur Laufzeit ein Fehler auf, erscheint eine Meldung am Bildschirm, in der eine entsprechende Fehlermeldung angezeigt wird. Diese Meldung lautet z. B. „Division durch Null“.

Programmtext

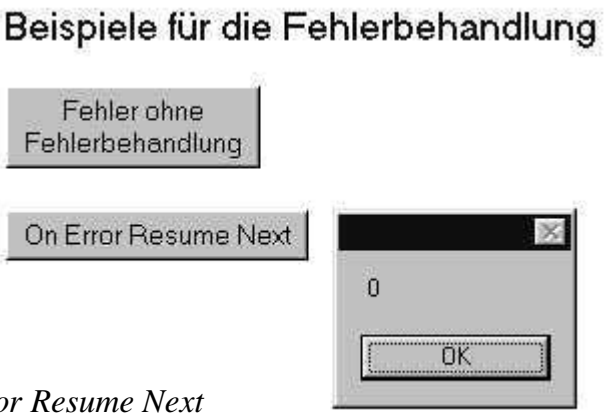
```
Dim Zahl1 As Integer
Dim Zahl2 As Integer
Dim Ergebnis As Double

Zahl1 = 7
Zahl2 = 0

Ergebnis = Zahl1 / Zahl2

‘ // Hier erfolgt der Abbruch und die Ausgabe der
‘ // Fehlermeldung

Msgbox Str$(Ergebnis)
```

Fehler ohne Fehlerbehandlung

Mit Hilfe der Anweisung „On Error Resume Next“ wird automatisch zur nächsten Anweisung verzweigt, ohne den Fehler zu beachten. Der Programmabbruch wird vermieden.

Programmtext

```
Dim Zahl1 As Integer
Dim Zahl2 As Integer
Dim Ergebnis As Double

On Error Resume Next

Zahl1 = 7
Zahl2 = 0

Ergebnis = Zahl1 / Zahl2
Msgbox Str$(Ergebnis)
```

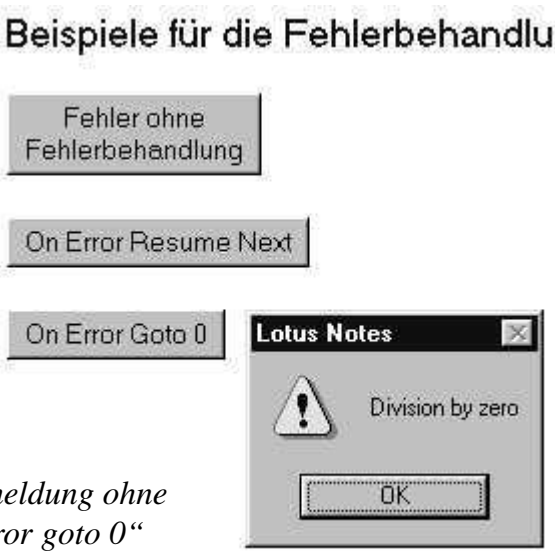


On Error Resume Next

Eine ähnliche Wirkung erzielt die Anweisung „On Error Goto 0“, wenn in der aufrufenden Prozedur die Fehlerbehandlung erfolgt.

Programmtext

```
Sub Click(Source As Button)
‘ ///
    ‘ // Mit dieser Anweisung wird die
    ‘ // Fehlermeldung verhindert.

    On Error Resume next

    Fehler3
‘ ///
End Sub

Sub Fehler3
‘ ///
    Dim Zahl1 As Integer
    Dim Zahl2 As Integer
    Dim Ergebnis As Double

    Zahl1 = 7
    Zahl2 = 0

    Ergebnis = Zahl1 / Zahl2

    ‘ // Hier wird die Prozedur Fehler verlassen.
    ‘ // Die Anweisung „Msgbox ...“ wird nicht
    ‘ // mehr ausgeführt
    Msgbox Str$(Ergebnis)
‘ ///
End Sub
```



Fehlermeldung ohne „On Error goto 0“

```
‘ // Fehlermeldung in der aufrufenden Prozedur!
```

Keine Fehlermeldung

```
‘ // Keine Fehlermeldung bei „On Error ...“
‘ // in der aufrufenden Prozedur.
‘ ///
```

```
    Dim Zahl1 As Integer
    Dim Zahl2 As Integer
    Dim Ergebnis As Double

    ‘ // Diese Anweisung gibt den
    ‘ // Fehler weiter an die
    ‘ // aufrufende Prozedur
    On Error Goto 0

    Zahl1 = 7
    Zahl2 = 0

    Ergebnis = Zahl1 / Zahl2
    Msgbox Str$(Ergebnis)
‘ ///
```

Die Unterdrückung von Fehlermeldungen ist in der Regel nicht ausreichend als Fehlerbehandungsmaßnahme. Eine weitere Variante des „On Error“-Befehls besteht deshalb darin, zu einer Sprungmarke zu verzweigen und dort genauer auf den Fehler zu reagieren. Um nach der Bearbeitung des Fehlers das Programm weiterführen zu können, bietet LotusScript eine Anweisung mit dem Namen „Resume“ an:

Resume [0 | **Next** | Sprungmarke]

Dabei bedeutet:

- **0**
  Rückkehr zu der Zeile, die den Fehler ausgelöst hat.
- **Next**
  Rückkehr zur Zeile, die der Programmzeile folgt, die den Fehler ausgelöst hat.
- **Sprungmarke**
  Das Programm wird an der angegebenen Sprungmarke fortgesetzt.

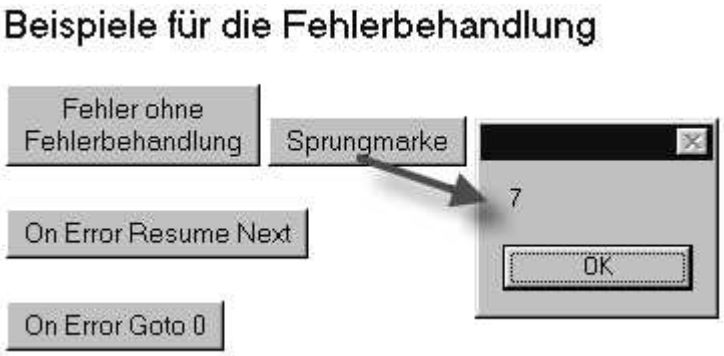
Nach der Resume-Anweisung wird der Fehler als erledigt angesehen und alle Fehlervariablen werden zurückgesetzt.

Die Anweisung „Resume 0“ kann zu einer Endlos-Schleife führen. Wenn die Fehlerursache nicht vor der Resume-Anweisung behoben wurde, löst die gleiche Programmzeile wieder einen Fehler aus!

Kehren wir zurück zu der eigentlichen Fehlerbehandlung. Die Variante, die noch fehlt, ist die Anweisung „On Error Goto Sprungmarke“. Bei dieser Variante wird jetzt eine Sprungmarke gesetzt, bei der die Fehlerbehandlung durchgeführt und dann über „Resume“ das Programm fortgesetzt wird.

Programmtext

```
Sub Click(Source as Button)
‘ ///
    Dim Zahl1 As Integer
    Dim Zahl2 As Integer
    Dim Ergebnis As Double

    ‘ // Hier wird zur Fehlerbehandlung
    ‘ // verzweigt
    On Error Goto Fehler

    Zahl1 = 7
    Zahl2 = 0

    Ergebnis = Zahl1 / Zahl2
    Msgbox Str$(Ergebnis)

    ‘ // Wenn kein Fehler auftritt,
    ‘ // wird das Programm hier
    ‘ // normal beendet
    Exit Sub

Fehler:
    Zahl2 = 1
    Resume 0
    Exit Sub

‘ ///
End Sub
```



Sprungmarke

Das Programm wird nach der Division durch Null zur Sprungmarke „Fehler“ umgeleitet. Der Variablen „Zahl2“ wird ein gültiger Wert zugewiesen und über „Resume 0“ wird die Division wiederholt. Sie führt jetzt zu einem gültigen Ergebnis, das in der Hinweisbox angezeigt wird.

Alle bisher genannten Beispiele zur Fehlerbehandlung beziehen sich auf alle Fehler, die nach der Anweisung „On error ...“ auftreten. Es ist aber auch möglich, auf einen Fehler ganz gezielt zu reagieren. Dazu kann der Fehlercode in der „On Error“-Anweisung verwendet werden. Außerdem bietet LotusScript eine Reihe von Fehlerfunktionen, die im Fehlerfalle automatisch gefüllt werden:

- **Erl-Funktion**
  Mit Hilfe dieser Funktion findet man die Programmzeilen-Nummer, in der der Fehler aufgetreten ist, als Integer-Wert.

  ```
  Print Erl

  ‘ // Druckt die Fehler-Zeilennummer
  ```

- **Err-Funktion**
  Die Err-Funktion gibt den Fehlercode des aufgetretenen Fehlers als Zahl vom Typ Integer zurück. Ist kein Fehler aufgetreten, so ist das Ergebnis der Funktion 0 (False).

  ```
  if Err = 1002 then ...
  ```

- **Err-Anweisung**
  Im Gegensatz zur Err-Funktion dient die Anweisung zur Festlegung eines eigenen Fehlercodes. Damit können z. B. auch Eingabefehler eines Benutzers im Rahmen der Fehlerbehandlung bearbeitet werden.

  ```
  Err = 4711
  ```

- **Error-Funktion**
  Liefert den Fehlertext zu einem Fehlercode.

  ```
  Msgbox Error(1002)
  ```

- **Error-Anweisung**
  Löst einen Fehler aus. Gibt man einen LotusScript Fehlercode an, wird dieser verwendet. Ein eigener Fehlercode kann ebenfalls gesetzt und dann ausgewertet werden.

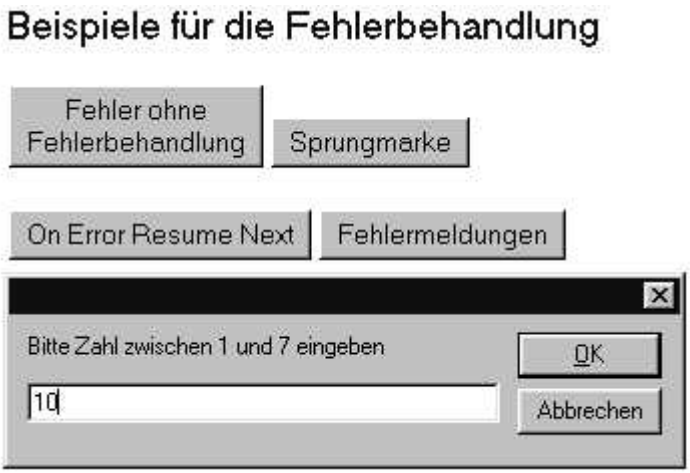
  ```
  Error 4711
  ```

Durch den Einsatz der Anweisungen und Funktionen zur Fehlerbehandlung kann man nun differenziert auf unterschiedliche Fehlersituationen reagieren.

Programmtext

```
Dim Zahl1 As Integer
Dim Zahl2 As Integer
Dim Ergebnis As Double

On Error Goto Fehler

NeueZahl:

Zahl1 = Val(Inputbox$(„Bitte Zahl“ & _
  „ zwischen 1 und 7 eingeben!“ & _
  „ 0 = Abbrechen:“))

If Zahl1 > 7 Then Error 9801
Zahl2 = 0

Ergebnis = Zahl1 / Zahl2
Msgbox Str$(Ergebnis)

‘ // Wenn kein Fehler auftritt,
‘ // wird das Programm hier
‘ // normal beendet
Exit Sub

Fehler:

If Err = 9801 Then
  Msgbox „Die Zahl darf nicht größer als 7 sein!“
  Resume NeueZahl
```

```
Else
  ‘ // Fehler beseitigen
  Zahl2 = 1
  Msgbox „Division durch 0 wurde korrigiert!“
End If

‘ // Fehlerzeile in der Statuszeile ausgeben
Print „Fehlerzeile im Programm:“; Erl

Msgbox „Es ist ein Fehler aufgetreten:“ & _
  Chr(10) & Err & „ “ & Error$
Resume 0
Exit Sub
```



Eingabe für Fehlerprüfung

Nach der Abfrage der Zahl erfolgt die Prüfung innerhalb der Prozedur. Ist die Zahl ungültig, wird ein Fehlercode gesetzt und eine Meldung ausgegeben.

Benutzerdefinierte Fehlermeldung

Am Ende der Fehlerbehandlung wird in jedem Fall der Fehler mit Code und Beschreibung gezeigt.

Ausgabe der Standardfunktionen

6.3.3.2 Fehlerbehandlung über eine spezielle Routine

Da die Bearbeitung von Fehlern in jeder Prozedur geregelt sein sollte, werden Fehlerbehandlungsanweisungen sehr häufig eingesetzt. Es bietet sich also an, die Behandlung von Fehlern zu standardisieren und in eigenständige Routinen auszulagern. Dazu ist zunächst zu regeln, wie Fehlermeldungen ausgegeben werden sollen. Die Ausgabe kann erfolgen:

- als Text in der Statuszeile
- als Nachricht in einer Meldung
- als Eintrag in einer Textdatei
- als Eintrag in die Datenbank „log.nsf“
- als Eintrag in eine Fehlerdatenbank
- als e-Mail

Je nach Ausgabeart müssen unterschiedliche Methoden zur Verarbeitung der Fehler eingesetzt werden. Um das Prinzip zu erläutern, werden wir hier zunächst die Ausgabe in die Datenbank „log.nsf“ beschreiben. Die Ausgabe über Meldungen und die Statuszeile wurde in den vorhergehenden Beispielen bereits verwendet.

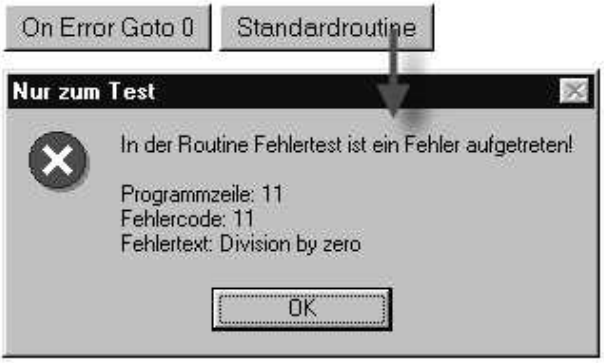
Der Programmtext für die Fehlerbehandlung wird in eine Prozedur mit dem Namen „Fehlerlog“ in eine Script-Bibliothek ausgelagert. In den einzelnen Prozeduren und Funktionen wird der Text für den Aufruf der Fehlerprozedur eingefügt und der Name der Prozedur angepasst. Für die Anzeige am Bildschirm wurde hier zusätzlich eine Meldung eingesetzt (das ist nicht immer erforderlich).

Programmtext

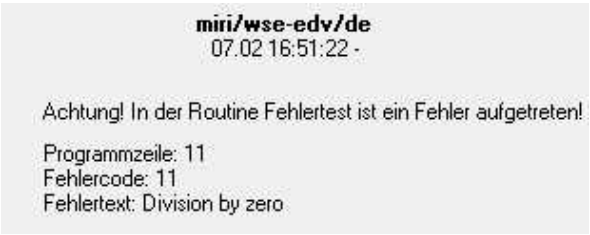
```
‘ // Beispiel für eine Prozedur,
‘ // die die Standard-Fehlerprozedur
‘ // aufruft.
‘ // Die fettgedruckten Zeilen
‘ // werden immer eingesetzt.

Sub Fehlertest
‘ ///
    On Error Goto Fehler
    Dim Zahl1 As Integer
    Dim Zahl2 As Integer
    Dim Ergebnis As Double

    Zahl1 = 7
    Zahl2 = 0

    Ergebnis = Zahl1 / Zahl2
    Msgbox Str$(Ergebnis)
    Exit Sub

Fehler:
    ‘ // Hier wird der Name der Prozedur
    ‘ // eingesetzt
    Fehlerlog Err, Erl, Error$, _
    „Fehlertest“
    Exit Sub
‘ ///
End Sub
```

```
Sub Fehlerlog(ErrNr As Integer, _
    ErrLine As Integer, ErrText As String, ErrProz As String)
‘ ///
    Dim ses As New NotesSession
    Dim db as NotesDatabase
    Dim dbOut As NotesDatabase
    Dim doc As NotesDocument
    Dim ErrMsg As String

    ErrMsg = „In der Routine“ & ErrProz & _
    „ist ein Fehler aufgetreten!“ & Chr(10) & _
    Chr(10) & _
    „Programmzeile:“ & ErrLine & Chr(10) & _
    „Fehlercode:“ & ErrNr & Chr(10) & _
    „Fehlertext:“ & ErrText
    ‘ // Hier können noch weitere Angaben folgen

    Msgbox ErrMsg, 16, „Nur zum Test“

    Set db = ses.CurrentDatabase
    Set dbOut = ses.getDatabase(db.Server, _
      „log.nsf”)
    Set doc = New NotesDocument(dbOut)

    If Not doc Is Nothing Then

      With doc
        .Form = „Events”
        .Server = db.Server
        .Events = „Fehler“
        Set doc.StartTime = _
        New NotesDateTime(Today)
```

```
        .Body = ErrMsg
        .save True, False
      End With

    End If
' ///
End Sub
```



Fehlerbehandlung über eine Standardroutine



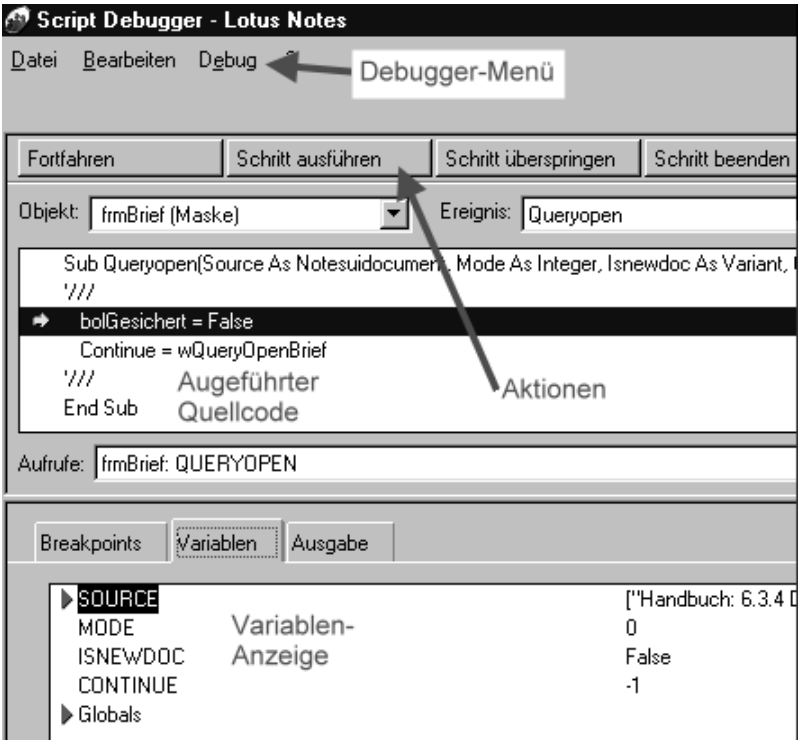
Fehlermeldung in der Datei „log.nsf“

Wenn die Datenbank „log.nsf“ als Empfänger der Fehlermeldungen verwendet wird, sind sicher noch weitere Informationen notwendig, die in der Fehlermeldung enthalten sein sollen. Solche Angaben sollten z. B. über den Benutzer, das Programmpaket und die benutzte Datenbank gemacht werden.

Diese Seite ist zur Zeit nicht belegt.

6.3.4 Der LotusScript-Debugger

Zur Entwicklungsumgebung gehört ein komfortabler Debugger. Das ist ein Programm, welches es ermöglicht, zur Laufzeit festzustellen, wie der genaue Programmablauf ist, welche Inhalte die Variablen haben und an welcher Stelle ein bestimmter Fehler auftritt. Der Debugger kann sowohl vom Designer als auch vom Notes-Client aus gestartet werden. Der Aufruf erfolgt über das Menü: „Datei / Extras / Debug LotusScript“. Er wird am Bildschirm aktiv, sobald die Script-Ausführung beginnt.



Das Debug-Fenster

Das Debug-Fenster enthält unterschiedliche Bereiche, über die der Programmablauf und die Anzeige gesteuert werden. Im Menü befindet sich eine Position „Debug“ mit folgenden Auswahlmöglichkeiten:

- Schritt ausführen, überspringen oder beenden
- Fortfahren
- Stopp
- Breakpoint festlegen, löschen, (de)aktivieren
- Alle Breakpoints löschen, (de)aktivieren
- Umschalten auf Breakpoints, Variablenanzeige oder Anzeige der Ausgabe

Darunter befinden sich Schaltflächen für die Aktionen:

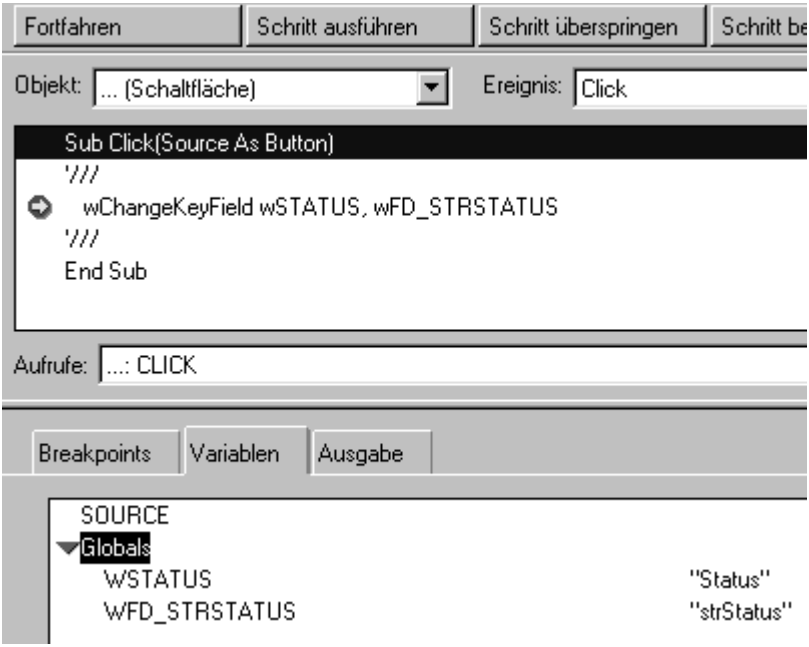
- Fortfahren
  Programmausführung fortsetzen, ohne schrittweise anzuhalten.
- Schritt ausführen
  Die nächste Programmanweisung ausführen.
- Schritt überspringen
  Aktuelle Zeile nicht ausführen, sondern mit der darauf folgenden Zeile weitermachen.
- Schritt beenden
  Beendet die aktuelle Funktion, Prozedur oder Schleife und setzt die Verarbeitung in der auf den Aufruf folgenden Zeile fort.
- Stopp
  Beendet nach einer Rückfrage den Debugger.

Über die Auswahlfenster „Objekt“ und „Ereignis“ kann zu anderen Programm-Modulen gesprungen werden, ohne die Position der Programmausführung zu verändern. Das dient vor allem dazu, Breakpoints zu setzen, also Haltepunkte, bei denen die Programmausführung gestoppt werden soll.

Im oberen Fenster wird der auszuführende Programmcode angezeigt. Die auszuführende Zeile ist farbig hinterlegt. Mit der Aktionsschaltfläche „Schritt ausführen“ wird die Programmzeile abgearbeitet. Wird die Schaltfläche „Fortfahren“ angeklickt, läuft das Programm ohne Anzeige im Debugger weiter bis zum nächsten Breakpoint oder – wenn keine Breakpoints mehr erreicht werden – bis zum nächsten Benutzerdialog bzw. bis zum Programmende.

Das untere Fenster kann drei unterschiedliche Informationsarten anzeigen:

- Breakpoints
- Variablen
- Ausgabe

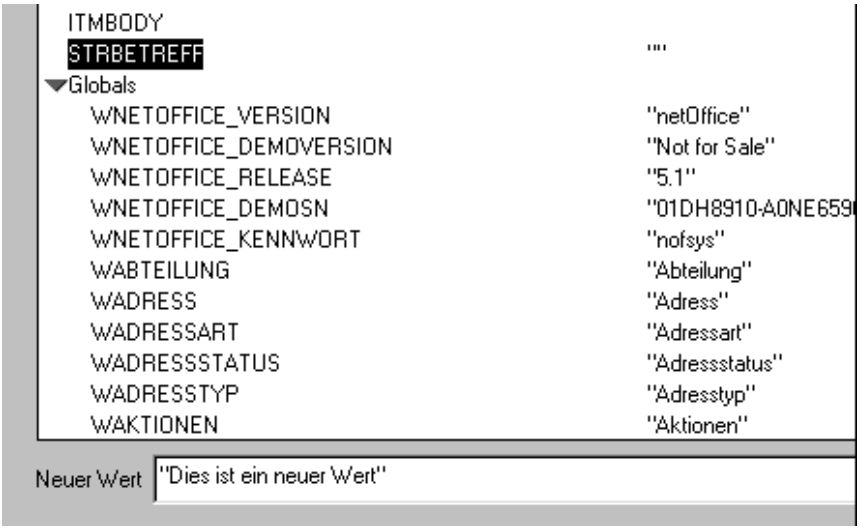


Anzeige im Debugger

Bei den Breakpoints wird die Prozedur / Funktion und die Zeile des jeweiligen Breakpoints angezeigt. Es handelt sich also um eine Liste der gesetzten Breakpoints.

In dem Ausgabefenster werden alle Print-Befehle des aktuellen Programms ausgeführt. Damit können z. B. Änderungen in Variablen protokolliert werden.

Im Fenster „Variablen“ schließlich wird der Status der einzelnen Variablen passend zu der ausgeführten Programmzeile angezeigt. Auch der Inhalt der Variablen ist zu sehen. Bei den definierten Objekten wird mit einem Twistie (einem kleinen Dreieck) gekennzeichnet, ob ein Objekt initialisiert ist oder nicht. Die Werte der Variablen können einfach geändert werden.



Wert einer Variablen verändern

Die zu ändernde Variable wird angeklickt und in dem Eingabefeld „Neuer Wert“, das am unteren Rand ist, kann jetzt der gewünschte Wert eingetragen werden. Damit ist es möglich, das

Programm weiterzutesten, wenn an einer Stelle ein falscher Variablenwert das Ergebnis verfälschen würde.

Leider ist der Debugger nicht in der Lage, das Script in Dialogmasken zu bearbeiten. Auch wenn der Code in externe Textdateien ausgelagert wurde, verweigert der Debugger seine Dienste. Hier empfiehlt es sich, bei Modulen, die neu geschrieben wurden und noch viele Fehlermöglichkeiten enthalten, den Code in eine Script-Bibliothek zu kopieren und ihn erst, wenn alle Fehler beseitigt sind, in die externe Datei zurückzuverlagern.

6.3.5 Klassen und Objekte

Den verschiedenen Basic-Dialekten wird üblicherweise nachgesagt, dass sie noch keine Objektorientierung kennen und eher einfach ausgelegt sind. Diese Aussage trifft heute in den wenigsten Fällen noch zu. Visual Basic von Microsoft verfügt über ein ausgefeiltes Objektmodell und auch LotusScript kennt schon seit längerem Klassen und Objekte. LotusScript unterstützt ein eigenes Modell, hat aber auch das Microsoft-Modell integriert. Der Entwickler kann also wählen, welches Modell er einsetzen will.

Die Welt der Objekte hat ihre eigene Logik, die für Entwickler, die aus dem prozeduralen Sprachumfeld kommen, eine gewisse Einarbeitung voraussetzt. Dabei stellt sich zunächst die Frage: Was bedeutet „Klasse“, und wie unterscheidet sich die Klasse vom „Objekt“. Einfacher gesagt, bedeutet Klasse eine allgemeine Beschreibung (wie etwa der Bauplan für ein Haus), und Objekt stellt das aus der Klasse erzeugte Produkt (also das Haus, das dem Bauplan entspricht) dar.

Damit man mit Objekten arbeiten kann, werden sie um Eigenschaften (z. B. Dachform, Grundriss usw.) und Methoden (z. B. „erstelleRaum“ erweitert. Damit können Programmteile dem Objekt zugeordnet und vor dem Rest des Programmes verborgen (gekapselt) werden. Es entstehen mit solchen Objekten Programme, die – richtiger Einsatz vorausgesetzt – weniger Fehler und wieder verwendbare Module enthalten.

Mai 2001 / 5. EL

6.3.5.1 Objekte

Voraussetzung für die Zusammenfassung von Programmtexten zu Modulen, die Zusammenhänge darstellen und Eigenschaften beschreiben, ist die Definition eines übergeordneten Begriffes. Solche Begriffe, die zur Kategorisierung von Funktionen und Eigenschaften geeignet sind, nennt man Objekte. Objekte sind sozusagen der Ausgangspunkt für die Beschreibung dessen, was entstehen und damit getan werden soll. Es ergeben sich zwei Richtungen, in die Objektbeschreibungen gehen:

- Objekteigenschaften
- Objektmethoden

Versucht man die Eigenschaften eines Objektes zu beschreiben, so erkennt man sehr schnell, dass es Objekthierarchien gibt, bei denen allgemeine Eigenschaften zu vielen und spezielle Eigenschaften zu wenigen Objekten passen. Wird z. B. „Tier“ als Objekt definiert, so passen zu diesem Objekt allgemeine Eigenschaften wie Farbe, Größe usw. Innerhalb der Tiere lassen sich wieder Gruppierungen als Objekte definieren, die über speziellere Eigenschaften verfügen (z. B. Vögel, Fische, Säugetiere usw.). Ähnlich den Eigenschaften gilt auch für Aktivitäten, dass es allgemeine und spezielle Tätigkeiten und Fähigkeiten gibt, die entweder auf alle Tiere, ganze Tiergruppen oder nur auf einzelne Tierarten zutreffen. Die Überlegung, dass man von einer allgemeinen Beschreibung von Eigenschaften und Methoden zu ihrer immer weitergehenden Spezialisierung kommen kann und damit eine Objekthierarchie beschreiben kann, ist die Grundlage der Objekttheorie bei der Programmierung. Auch hier werden zunächst Objekte gebildet, die durch wenige wichtige Eigenschaf-

ten gekennzeichnet sind (z. B. Adressen, Artikel usw.). Dazu kommen Tätigkeiten (Methoden) die im Zusammenhang mit diesen Objekten stehen (Erstellen, Löschen usw.). Bei der Beschreibung von Objekten wird versucht, alles, was mit ihnen zusammenhängt, in ein Modell zu fassen, und Programmfunktionen nur in Anbindung an ein Objekt zu beschreiben. Es wird also nicht eine Abfolge von Programmschritten definiert, deren Wiederholung irgendwie organisiert wird, sondern es wird versucht, Programmschritte an ein Objekt zu binden, und dieses Objekt immer dann einzusetzen, wenn der Programmzusammenhang dies erfordert. Als Ergebnis dieser Vorgehensweise soll der Programmcode zu unabhängigen Modulen zusammengefasst werden, der über Schnittstellen mit anderen Modulen kommuniziert. Dabei bestehen keine Beziehungen zwischen den Programmabläufen innerhalb der verschiedenen Module. Man spricht hier von einer Kapselung des Programmcodes.

6.3.5.2 Klassen

Da das Ziel der Objektbildung darin besteht, sie immer da einzusetzen, wo ihre Funktionen erforderlich sind, wäre es nicht sinnvoll, jedes Mal eine Kopie ihres Programmcodes einzusetzen. Es wird vielmehr eine einmalige Definition eines Objektes vorgenommen und an einer Stelle hinterlegt. Diese Definition bezeichnet man als Klasse. Aus der Klasse – also der Objektdefinition – werden so viele Objekte im Arbeitsspeicher erzeugt, wie es ein Programm erfordert. Alle diese Objekte verfügen, da sie von der gleichen Klasse abstammen, über die gleichen Eigenschaften und Methoden. Die Klasse wird also an einer Stelle definiert. Sobald die Klasse existiert, können überall im Programm Objekte aus dieser Klasse generiert und verwendet werden.

6.3.5.3 Ereignisse

Ein Objekt als Programmelement ist noch zu grob geschnitten, um alle Erfordernisse abzudecken, die an die moderne Programmierung gestellt werden. Zu den Eigenschaften und Methoden kommen deshalb bei bestimmten Objekten noch Ereignisse hinzu. Diese Ereignisse werden z. B. durch Benutzer oder andere Programme ausgelöst und müssen vom Objekt in Form von Programmfunktionen beantwortet werden. Typische Ereignisse sind:

- Click
- Entering
- Initialize
- Postopen
- Querysave
- usw.

6.3.5.4 Benutzerdefinierte Klassen

Aus dem Zusammenspiel von Objekten, Methoden und Ereignissen entsteht in der objektorientierten Programmierung eine Anwendung. Zur Erstellung von Objekten können Basisklassen und abgeleitete Klassen definiert werden. Wie bereits ausgeführt, können allgemeine Objektbeschreibungen dazu dienen, Kategorien und Gattungen zu beschreiben. Basisklassen sind also die Grundlage, auf denen dann spezielle, abgeleitete Klassen aufbauen. Nimmt man z. B. eine Klasse „Adressen“, so lassen sich davon Firmenadressen, Personenadressen, Lieferadressen usw. ableiten. Eine abgeleitete Klasse übernimmt automatisch alle Eigenschaf-

ten und Methoden der Basisklasse. Zusätzlich können in der abgeleiteten Klasse neue Methoden und Eigenschaften definiert werden.

Die Syntax einer Basisklasse lautet:

[Public | Private] Class ***Name der Klasse***

Definitionsteil der Klasse

End Class

Zu Beginn der Definitionen innerhalb der Klasse sind direkt zwei Besonderheiten zu vermerken. Die erste ist, dass Variablen ohne das Schlüsselwort „Dim" definiert werden. Die zweite ist die, dass an dieser Stelle keine Zuweisungen zu den Variablen vorgenommen werden können. Die Variablen am Anfang der Klasse sind Objektvariablen und können entweder als „Public" oder als „Private" definiert werden. Public-Variablen sind von außerhalb des Objektes gültig und können von außen und von Funktionen bzw. Prozeduren innerhalb des Objektes mit Werten versehen werden. Wird nichts angegeben oder „Private" so sind diese Variablen nur innerhalb des Objektes verfügbar. Nach den Objektvariablen werden Eigenschaften und Methoden definiert. Die Methoden stellen sich als Prozeduren und Funktionen dar. Für die Bearbeitung von Eigenschaften gibt es eine weitere Besonderheit: das Funktionspaar „Property Set" und „Property Get". Mit diesen Funktionen können Eigenschaften mit Inhalt versehen und abgefragt werden, die innerhalb der Klasse als „Private" definiert sind und damit außerhalb der Klasse nicht existieren. Der Vorteil dieser Methodik besteht darin, dass z. B. Gültigkeitsprüfungen möglich sind, bevor eine Variable einen Wert erhält, dass die Variable von außen nur „gelesen" oder nur „geschrieben" werden kann.

Das Bild einer Klasse sieht nun folgendermaßen aus:

Programmtext

```
Class Name
    Objektvariable

    Sub Name
      Prozedurtext
    End Sub

    Function Name
      Text der Funktion
    End Function

    Property Set Name
      Text für Eigenschaft
    End Property

    Property Get Name
      Text für Eigenschaft
    end Property

End Class
```

6.3.5.4.1 Die Prozeduren „New“ und „Delete“

Die Klasse in LotusScript kennt zwei spezielle Prozeduren, die bei der Erzeugung bzw. bei der Zerstörung eines Objektes aufgerufen werden. Die New-Prozedur entspricht dem Konstruktor in anderen Programmiersprachen und dient zur Initialisierung des Objektes. Hier werden z. B. die Anfangswerte der Objektvariablen gesetzt. Dabei können auch Parameter verwendet werden, die an die New-Prozedur übergeben wurden. Umgekehrt kann in der Delete-Prozedur hinterlegt werden, was geschehen soll, bevor das Objekt aus dem Arbeitsspeicher entfernt wird.

Programmtext

```
Class Mitarbeiter
‘ ///
    Name as string
    Vorname as string
    Status as string

    Sub New(strName as string, _
      strVorname as string, _
      strStatus as string)
    ‘ ///
      Name = strName
      Vorname = strVorname
      Status = strStatus
    End Sub
```

```
‘ /// Weitere Prozeduren und Funktionen

Sub Delete
  Print „Mitarbeiter entfernt“
End Sub
```

6.3.5.4.2 Objekte löschen und Speicherverwaltung

Die Speicherverwaltung unter LotusScript erfolgt automatisch. Es muss also nicht darauf geachtet werden, wo ein Speicher für ein Objekt reserviert und wann er wieder freigegeben wird. Wenn es einmal erforderlich sein sollte, so kann ein Objekt mit Hilfe einer Anweisung aus dem Speicher entfernt werden. Die Anweisung heißt „Delete“ und hat folgende Syntax:

- **Delete**
  Name des Objektes

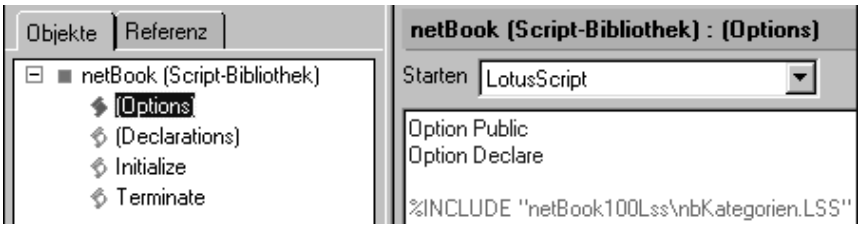
Eine weitere Methode, ein Objekt aus dem Speicher zu entfernen, besteht darin, seinen Wert auf „NOTHING“ zu setzen.

Programmtext

```
Dim Objektxy as new UserObjekt
Dim Objektab as new UserObjekt

‘ /// Programmtext
Delete Objektab
Objektxy = Nothing
```

6.3.5.4.3 Klassen definieren und Objekte verwenden

Klassen können bei jedem Ereignis, das LotusScript unterstützt, definiert werden. Damit erkauft man sich allerdings den Nachteil, dass solche Klassen nur begrenzt gültig und schwer auffindbar sind. Besser ist deshalb die Methode, Klassen in dem Abschnitt „Script-Bibliotheken“ der Ressourcen zu beschreiben.

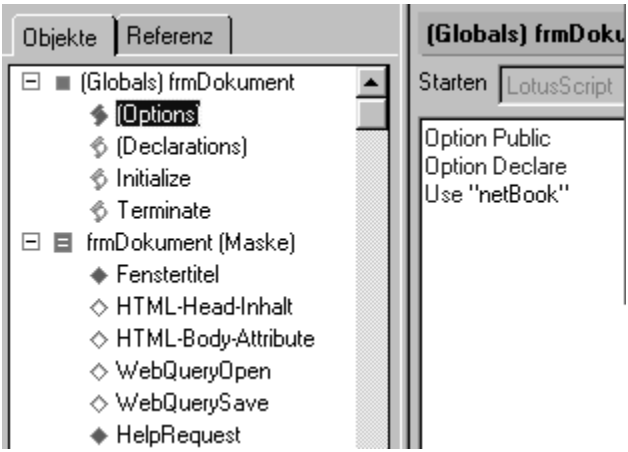


Script-Bibliothek

Legt man eine solche Script-Bibliothek an, so verfügt sie unter anderem über das Ereignis „(Declarations)“. In diesem Bereich werden Klassendefinitionen erstellt. Will man diese Script-Bibliothek verwenden, so wird dazu die Anweisung „Use“ verwendet. Die Anweisung hat die Syntax:

- **Use**
  “Name der Script-Bibliothek“

Die Use-Anweisung wird in dem Optionsereignis des Moduls eingesetzt, bei dem Teile der Script-Bibliothek verwendet werden.



Die Use-Anweisung

In den Deklarationen kann jetzt die Klasse mit ihren Elementen definiert werden.

Programmtext

```
Class wAttachment
‘ ///
    Path As String
    Form As String
    Icon As Integer

    Sub New(strPath As String)
    ‘ ///
      Me.Path = strPath
    ‘ ///
    End Sub

      Property Get SourcePath As String
    ‘ ///
      Me.getPfad
      SourcePath = Me.Path
    ‘ ///
    End Property

    Sub getPfad
    ‘ ///
    ‘ /// Programmtext
    ‘ ///
    End Sub

    Sub setPfad
    ‘ ///
    ‘ /// Programmtext
    ‘ ///
    End Sub
```

```
    Sub attachFiles(strArt As String)
    ‘ ///
    ‘ /// Programmtext
    ‘ ///
    End Sub

‘ ///
End Class
```

In einer Ansicht, die die oben definierte Klasse als Objekt einsetzt, kann folgender Text stehen:

```
Dim att as New wAttachment(„C:\Eigene Dateien“)

att.attachFiles(„Parameter“)
```

Im Programmtext werden die Methoden und Eigenschaften des Objektes mit der Syntax:

Objektname.Methode

bzw.

Objektname.Eigenschaft

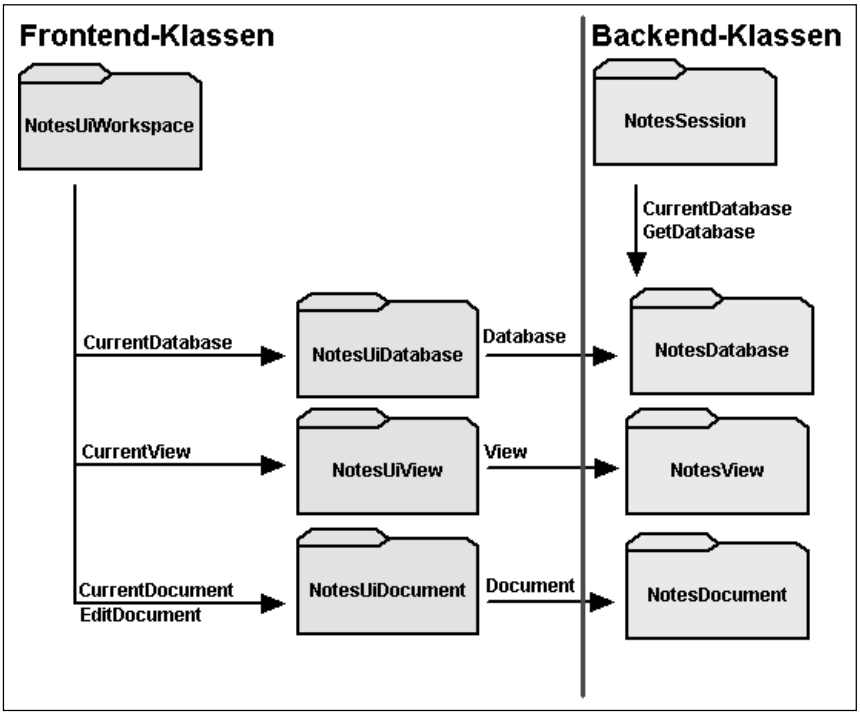
angesprochen. In der Klassendefinition werden Methoden und Eigenschaften als lokale Variable behandelt. Sie können, um Verwechslungen auszuschließen, mit dem Objekt „Me“ verbunden werden. Das Objekt „Me“ steht stellvertretend für den Namen der aktuellen Klasse.

6.3.6 Notes-Klassen

Neben den benutzerdefinierten Klassen liefert Lotus ein ganzes Bündel an Klassen im Sprachumfang mit, die dazu dienen, auf die verschiedenen Elemente des Domino-Systems zuzugreifen. Diese Klassen werden in zwei große Gruppen eingeteilt:

- Frontend-Klassen
- Backend-Klassen

Die Frontend-Klassen beschreiben Elemente, die der Benutzer sieht, also die Benutzeroberfläche. Da dafür der englische Begriff „User Interface“ lautet, haben diese Klassen die Abkürzung „Ui“ in ihrem Namen (NotesUiDocument, NotesUiWorkspace usw.). Die Backend-Klasse beschäftigt sich mit Elementen, die bereits gespeichert wurden. Die einzelnen Klassen sind miteinander verbunden und können oft nur dann verwendet werden, wenn ein Objekt der entsprechenden Oberklasse verfügbar ist. So enthalten beispielsweise Datenbanken die Ansichten und Dokumente die Felder. Felder können erst dann als Objekt angesprochen werden, wenn ein entsprechendes Dokumentenobjekt verfügbar ist.



Auszug aus den Notes-Klassen

Die Klassen dieses Domino-Objekt-Modells werden ständig erweitert und an die Anforderungen des Gesamtproduktes angepasst. Es muss sich dabei nicht unbedingt um neue Klassen handeln. Auch die bestehenden Klassen werden ständig weiterentwickelt und mit neuen Eigenschaften und Methoden versehen.

6.3.6.1 Frontend-Klassen

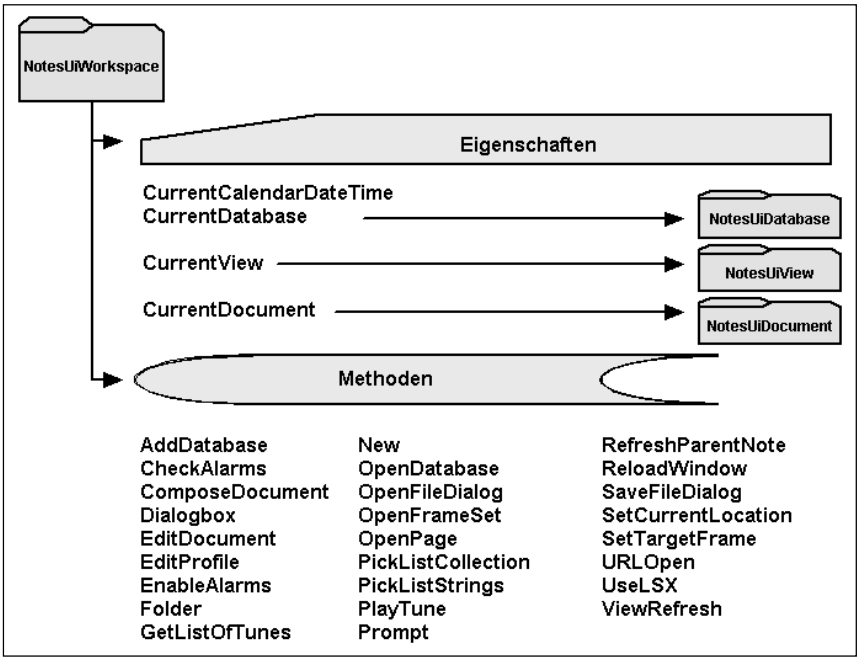
In den Frontend-Klassen sind Objekte zusammengefasst, die vom Benutzer geöffnet wurden. Das Grundelement für die Frontend-Klassen stellt die Klasse „NotesUiWorkspace“ dar. Es handelt sich dabei um Definitionen der aktuellen Arbeitsumgebung des Benutzers. Von NotesUiWorkspace aus kann dann auf die gerade verwendete Datenbank, die aktuelle Ansicht und das zur Zeit geöffnete Dokument zugegriffen werden. Als Frontend-Klassen sind zur Zeit verfügbar:

- NotesUiWorkspace
- NotesUiDatabase
- NotesUiDocument
- Navigator
- Field
- Button

Dabei verfügen die Klassen Navigator, Field und Button lediglich über Ereignisse, die für die Ausführung von LotusScript-Modulen verwendet werden können.

6.3.6.1.1 Die Klasse NotesUiWorkspace

Aus den Frontend-Klassen ist die Klasse NotesUiWorkspace diejenige mit der größten Bedeutung für die Programmierung. Hier sind die Methoden zur Anzeige und Bearbeitung von Dokumenten, Auswahllisten und Dialogen hinterlegt.



Die Klasse NotesUiWorkspace

Die Definition eines NotesUiWorkspace-Objektes erfolgt in einer Dim-Anweisung und unter Verwendung der New-Methode:

Programmtext

```
Dim ws as NotesUiWorkspace
Set ws = new NotesUiWorkspace

,/// oder

Dim ws as new NotesUiWorkspace
```

Damit stehen die Eigenschaften und Methoden der Klasse „NotesUiWorkspace“ für das Objekt „ws“ zur Verfügung. Als Eigenschaften enthält das Objekt:

- **CurrentCalendarDateTime**
  Diese Eigenschaft kann nur gelesen werden und enthält das aktuelle lokale Datum und die entsprechende Uhrzeit.
  Datum = NotesUiWorkspace.CurrentCalendarDateTime
- **CurrentDatabase**
  Mit Hilfe dieser Eigenschaft kann ein Objekt der Klasse NotesUiDatabase mit den entsprechenden Eigenschaften und Methoden angelegt werden.
  Set NotesUiDatabase = NotesUiWorkspace.CurrentDatabase

- **CurrentView**
  Über diese Eigenschaft wird ein Objekt für die gerade geöffnete Ansicht aus der Klasse NotesUiView erstellt.
  Set NotesUiView = NotesUiWorkspace.CurrentView

- **CurrentDocument**
  Der Zugriff auf das aktuell geöffnete Dokument wird häufig gebraucht, um auf Aktionen des Benutzers reagieren zu können. Dabei können z. B. Feldinhalte gelesen, geändert oder für eine Anzeige verwendet werden.
  Set NotesUiDocument = NotesUiWorkspace.CurrentDocument

Neben den Eigenschaften verfügt die Klasse NotesUiWorkspace auch über eine Reihe von wichtigen Methoden zur Benutzerführung und die Dokumentenbearbeitung:

- **AddDatabase**
  fügt eine Datenbank zum Register der Arbeitsoberfläche hinzu und setzt den Cursor auf das dazugehörige Icon.
  Call NotesUiWorkspace.AddDatabase (Server-Name, Datei-Name)

- **CheckAlarms**
  prüft den Alarm-Prozess, ob ein Alarm in der Mail-Datenbank vorliegt.
  *Call NotesUiWorkspace.*CheckAlarms

- **ComposeDocument**
  legt ein neues Dokument in der angegebenen Datenbank mit der angegebenen Maske an und öffnet es zur Bearbeitung.
  Set NotesUiDocument =
  NotesUiWorkspace.ComposeDocument
  ([Server-Name
  [, Dateiname [, Maske [, Fensterbreite
  [, Fensterhöhe

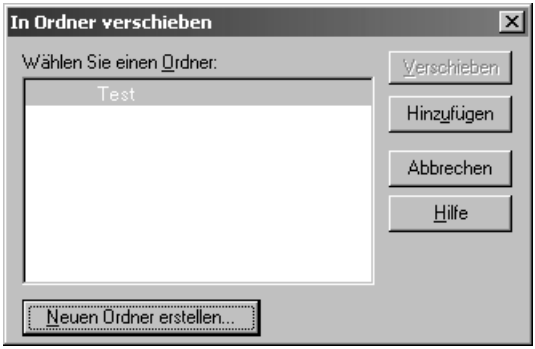
- **Dialogbox**
  öffnet ein Dialog-Fenster und zeigt eine Maske an. Der Benutzer kann z. B. mit „OK“ oder „Abbruch“ regieren. Diese Methode wird sehr häufig eingesetzt und bietet durch den Feldaustausch mit einem darunter liegenden Dokument viele interessante Möglichkeiten.
  True oder False = NotesUiWorkspace.Dialogbox (Maske
  [, horizontal anpassen [, vertikal anpassen
  [, kein Abbruch möglich
  [, keine Feldänderungen zulassen
  [, nur Lesen erlaubt [, Titel [, Dokumentenobjekt
  [, Größe an Tabelle anpassen
  [, keine Schaltflächen anzeigen]]]]]]]]])

- **EditDocument**
  wird dazu verwendet, ein Dokument zu öffnen und zu bearbeiten.
  Set NotesUiDocument =
  NotesUiWorkspace.EditDocument ([Modus
  [, Dokumentobjekt [, Dokument nur Lesen
  [, Ziel im Dokument]]])

- **EditProfile**
  Profildokumente sind spezielle Dokumente, die in Ansichten nicht angezeigt werden und die z. B. benutzerbezogene Daten enthalten können. Mit EditProfile werden solche Dokumente angelegt und bearbeitet.
  Set NotesUiDocument =
  NotesUiWorkspace.EditProfile (Name des Profildokumentes [, Notesbenutzer])

- **EnableAlarms**
  startet oder beendet den Alarm-Prozess.
  True oder False =
  NotesUiWorkspace.EnableAlarms (True oder False)

- **Folder**
  öffnet das „Verschieben in Ordner"-Dialogfenster von Notes und ermöglicht das Verschieben des aktuellen Dokuments in einen Ordner.



Verschieben in Ordner

Variantvariable = NotesUiWorkspace.Folder ([Name des Ordners
[, verschieben]])

- **GetListOfTunes**
  ermittelt die verfügbaren System-Klänge und stellt sie in ein Feld mit Mehrfachwerten.
  Liste der Systemklänge =
  NotesUiWorkspace.GetListOfTunes

- **New**
  erstellt ein Objekt aus der Klasse NotesUiWorkspace.
  Set Name des Arbeitsbereiches = New NotesUiWorkspace

- **OpenDatabase**
  öffnet eine Datenbank und zeigt die angegebene Ansicht an. Wird ein Schlüsselwert angegeben, so wird das passende Dokument markiert und die Ansicht bis zu diesem Dokument bewegt.
  Call NotesUiWorkspace.OpenDatabase (Server-Name, Datei-Name, Ansicht, Schlüsselwert, neues Fenster, temporär)

- **OpenFileDialog**
  öffnet einen „Datei-Öffnen"-Dialog des Windows-Systems.
  Datei-Namen =
  NotesUiWorkspace.OpenFileDialog (Mehrfachauswahl [, Titel [, Filter [, Ausgangsdateiverzeichnis [, Vorschlag für den Datei-Namen]]]])

- **OpenFrameSet**
  öffnet eine Rahmengruppe im aktuellen Zielrahmen.
  Call NotesUiWorkspace.OpenFrameSet (Name der Rahmengruppe)

- **OpenPage**
  öffnet eine Seite im aktuellen Zielrahmen.
  Call NotesUiWorkspace.OpenPage (Name der Seite)

- **PickListCollection**
  zeigt ein Dialogfenster an, das eine angegebene Ansicht enthält. Der Benutzer kann ein oder mehrere Dokumente auswählen. Das Programm erhält ein Objekt der Klasse NotesDocumentCollection zurück, das die gewählten Dokumente enthält.
  Set NotesDocumentCollection =
  NotesUiWorkspace.PickListCollection (Art der Anzeige, Mehrfachauswahl [, Server-Name] [, Datei-Name] [, Ansicht] [, Titel] [, Hinweis zur Auswahl] [, Kategorie als Filter])

- **PickListStrings**
  zeigt einen Dialog wie PicklistCollection, liefert aber ein Array vom Typ „String“ zurück, das die Werte aus der angegebenen Spalte der gewählten Dokumente enthält.
  Werteliste = NotesUiWorkspace.PickListStrings (Art der Anzeige, Mehrfachauswahl [, Server-Name] [, Datei-Name] [, Ansicht] [, Titel] [, Hinweis zur Auswahl] [, Spaltennummer])

- **PlayTune**
  spielt eine Klangdatei ab.
  Call NotesUiWorkspace.PlayTune (Name des Klangs)

- **Prompt**
  erlaubt die Anzeige einer Liste von Werten, aus der ein Benutzer einen oder mehrere Einträge auswählen kann. Das Programm erhält die gewählten Werte in Form einer Variant-Variablen zurück, die ein String-Array enthält.
  Werteliste = NotesUiWorkspace.Prompt (Art der Anzeige, Titel, Hinweis zur Auswahl oder Eingabe [, Vorgabewert] [, Auswahlwerte])

- **RefreshParentNote**
  aktualisiert das Dokument, das in einer Dialogbox angezeigt wurde.
  Call NotesUiWorkspace.RefreshParentNote

- **ReloadWindow**
  lädt den Inhalt des aktuellen Fensters neu.
  Call NotesUiWorkspace.ReloadWindow

- **SaveFileDialog**
  öffnet einen „Datei-Speichern"-Dialog des Windows-Systems.
  Werteliste =
  NotesUiWorkspace.SaveFileDialog (Mehrfachauswahl [, Titel] [, Startverzeichnis] [, Vorschlag für den Datei-Namen])

- **SetCurrentLocation**
  ändert die Arbeitsumgebung des Benutzers.
  Call NotesUiWorkspace.SetCurrentLocation (Name der Arbeitsumgebung)

- **SetTargetFrame**
  wird dazu verwendet, den Zielrahmen für die Aktionen OpenView, OpenPage usw. zu ändern.
  Call NotesUiWorkspace.SetTargetFrame (Name des Rahmens)

- **URLOpen**
  lädt eine Internet-Seite mit der angegebenen Internet-Adresse.
  Call NotesUiWorkspace.URLOpen (Intenet-Adresse, vom Server laden, Folgeadressen der Web-Seite sichern, Zeichensatz, Name des Server-Benutzers, Kennwort, Name des Proxy-Server-Benutzers, Kennwort, Browser)

- **UseLSX**
  lädt ein Programm-Modul.
  Call NotesUiWorkspace.UseLSX (Name der DLL-Datei)
- **ViewRefresh**
  aktualisiert die gerade geöffnete Ansicht.
  Call NotesUiWorkspace.ViewRefresh

Über das NotesUiWorkspace-Objekt kann auf die Objekte der Klassen NotesUiDatabase, NotesUiView und NotesUiDocument zugegriffen werden.

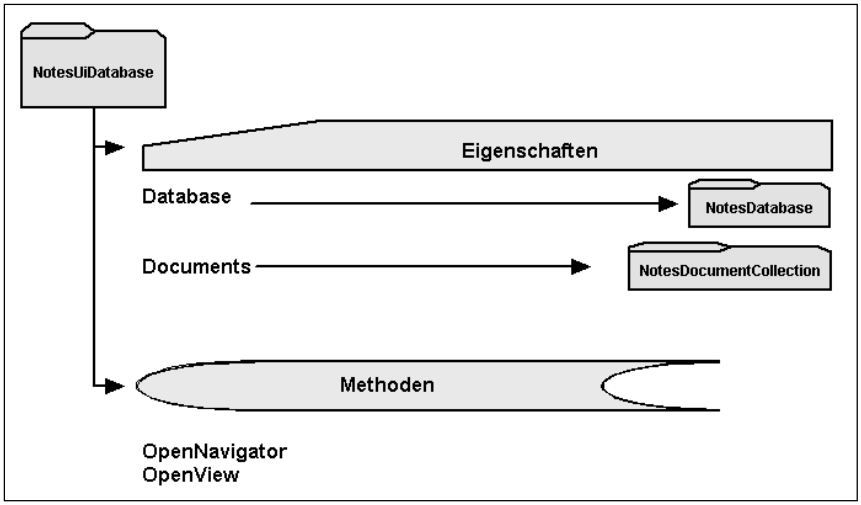
Programmtext

```
,/// Workspace-Objekt erzeugen
Dim ws as New NotesUiWorkspace
Dim uidb as NotesUiDatabase
Dim uivw as NotesView
Dim uidoc as NotesUiDocument

,/// Datenbankobjekt erzeugen
Set uidb = ws.CurrentDatabase

,/// Ansichtsobjekt erzeugen
Set uivw = ws.CurrentView

,/// Objekt für das aktuelle Dokument erzeugen
Set uidoc = ws.CurrentDocument
```

Für die neuen Objekte stehen jetzt deren Eigenschaften und Methoden zur Verfügung.

6.3.6.1.2 Die Klasse NotesUiDatabase

Die Frontend-Klasse der Datenbank hat nur wenige Eigenschaften und Methoden.



Die Klasse NotesUiDatabase

Dadurch wird sie relativ selten benötigt. Ihre Eigenschaften sind:

- **Database**
  enthält ein Objekt der Backend-Klasse NotesDatabase.
  Set NotesDatabase = NotesUiDatabase.Database

- **Documents**
  enthält alle Dokumente der aktuellen Datenbank als Objekte der Klasse NotesDocument in Form einer Dokumentensammlung, einem Objekt der Klasse NotesDocumentCollection.
  Set NotesDocumentCollection = NotesUiDatabase.Documents

Programmtext

```
Dim ws as New NotesUiWorkspace
Dim uidb as NotesUiDatabase
Dim db as NotesDatabase

Set uidb = ws.CurrentDatabase

,/// Datenbankobjekt ermitteln
Set db = uidb.Database
```

Die Klasse NotesUiDatabase verfügt über die zwei Methoden:

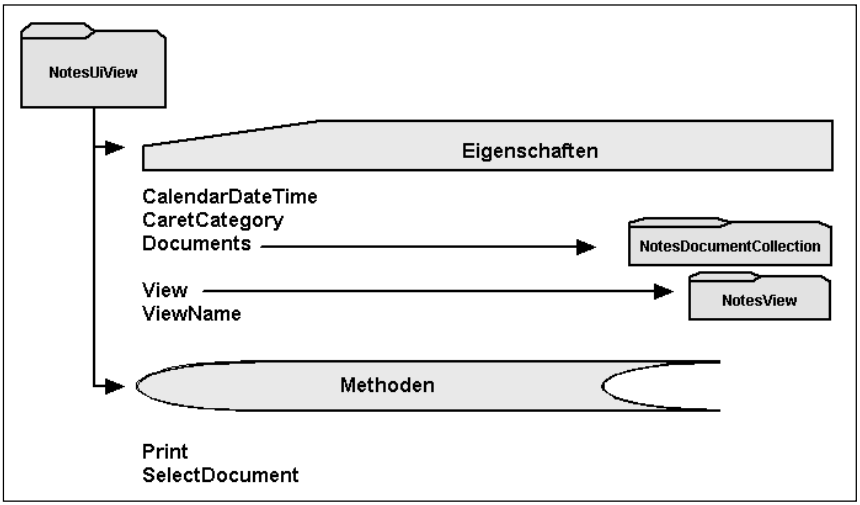
- **OpenNavigator**
  öffnet einen Navigator.
  Call NotesUiDatabase.OpenNavigator (Name des Navigators [, eigenes Fenster])
- **OpenView**
  öffnet eine Ansicht der aktuellen Datenbank.
  Call NotesUiDatabase.OpenView (Name der Ansicht [, Schlüsselwert] [, eigenes Fenster])

Programmtext

```
Dim ws as New NotesUiWorkspace
Dim uidb as NotesUiDatabase

Set uidb = ws.CurrentDatabase
call uidb.OpenView („Meeting“)

oder

call uidb.OpenNavigator („Marketing“)
```

6.3.6.1.3 Die Klasse NotesUiView

Die Klasse NotesUiView stellt ein Objekt zur Verfügung, das die gerade verwendete Ansicht enthält.



Die Klasse NotesUiView

Neben den Dokumenten, die über diese Ansicht angezeigt werden, enthält das Objekt auch die Möglichkeit, auf die Backend-Klasse „NotesView“ zuzugreifen.

- **CalendarDateTime**
  Datum und Zeit des aktivierten Bereiches in einer Kalender-Ansicht.
  Datum = NotesUiView.CalendarDateTime

- **CaretCategory**
  Inhalt des Kategorie-Feldes der aktuellen Ansicht (setzt voraus, dass diese Ansicht in der ersten Spalte kategorisiert ist).
  Werteliste = NotesUiView.CaretCategory

- **Documents**
  Dokumentenobjekte der geöffneten Ansicht.
  Set NotesDocumentCollection = NotesUiView.Documents

- **View**
  Referenz auf das NotesView-Objekt der Backend-Klasse
  Set NotesView = NotesUiView.View

- **ViewName**
  Name der aktuellen Ansicht.
  Name der Ansicht = NotesUiView.ViewName

Programmtext

```
Dim ws as New NotesUiWorkspace
Dim uivw as NotesUiView
Dim col as NotesDocumentCollection
Dim vw as NotesView
Dim doc as NotesDocument
Dim varFarben as Variant
Dim strName as String

,/// NotesUiView-Objekt erstellen
Set uivw = ws.CurrentView

,/// Kategoriewerte aus dem aktuellen Dokument übernehmen
varFarben = uivw.CaretCategory
```

```
,/// Einzelne Werte in der Statuszeile ausgeben
Forall Farbe in varFarben
    Print Farbe
End Forall

,/// Objekt für Dokumentensammlung ermitteln
Set col = uivw.Documents

,/// Anzahl der enthaltenen Dokumente als Meldung anzeigen
if col.count > 0 then
    msgbox „Die Ansicht enthält „ & str$ (col.count) & _
      „ Dokumente.“
End if

,/// Ansichtsobjekt erstellen
Set vw = uivw.View

,/// Dokument-Objekt erstellen
Set doc = vw.GetDocumentByKey („Grün“)

,/// Wenn ein Dokument mit dem Schlüssel „Grün“ vorhanden ist, ,///
das Feld „Subject“ in der Statuszeile ausgeben
if not doc is Nothing then
    print doc.Subject(0)
End if

,/// Name der aktuellen Ansicht ermitteln
strName = uivw.ViewName
```

Als Methoden bietet die NotesUiView-Klasse das Drucken und die Auswahl eines Dokumentes an:

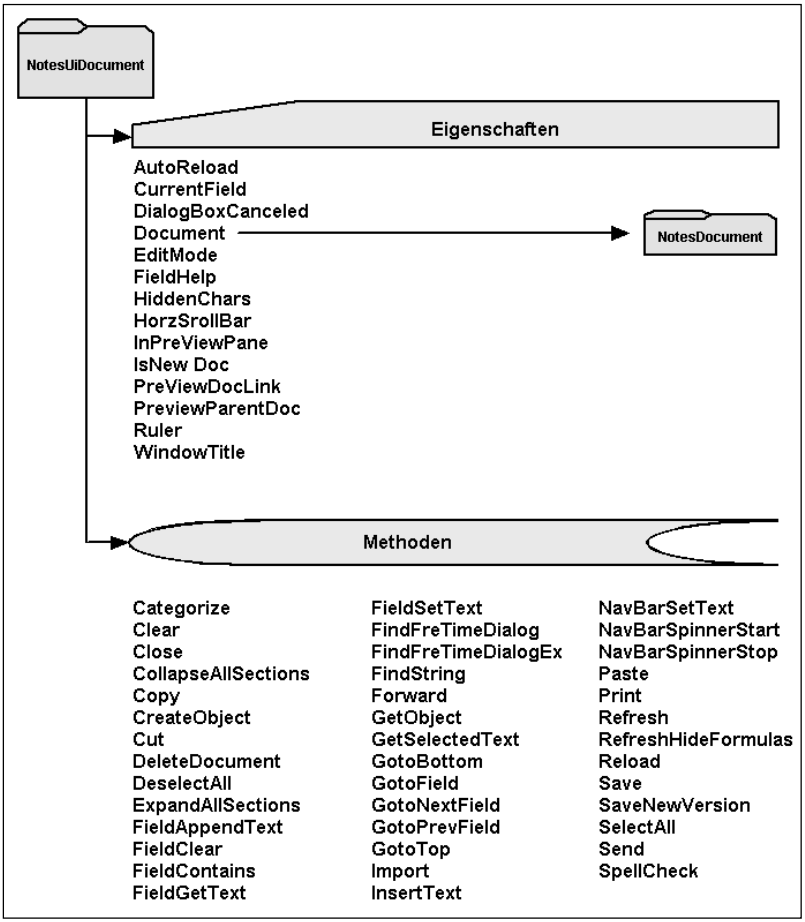
- **Print**
  druckt die markierten Dokumente oder die aktuelle Ansicht.
- **SelectDocument**
  wählt ein Dokument aus, dessen Objekt als Parameter angegeben wird.

Programmtext

```
Dim ws as New NotesUiWorkspace
Dim uivw as NotesUiView

,/// Objekt für die aktuelle Ansicht erstellen
Set uivw = ws.CurrentView

,/// Ansicht drucken
uivw.Print
```

6.3.6.1.4 Die Klasse NotesUiDocument

Die Klasse NotesUiDocument ist eine der wichtigsten aus den Frontend-Klassen. Fast immer, wenn Feldinhalte eines geöffneten Dokumentes in LotusScript-Modulen verarbeitet werden, wird auf diese Klasse zurückgegriffen.



Die Klasse NotesUiDocument

Aus diesem Grund enthält diese Klasse eine große Anzahl von Eigenschaften und Methoden. Die Eigenschaften sind:

- **AutoReload**
  Diese Eigenschaft regelt, ob Felder (die nicht den Typ „RichText" haben) automatisch im aktuellen Dokument angepasst werden, wenn das dazugehörige Backend-Dokument geändert wird.
  True oder False = NotesUiDocument.AutoReload
  NotesUiDocument.AutoReload = True oder False

- **CurrentField**
  liefert den Namen des Feldes, in dem der Cursor gerade steht.
  Name des Feldes = NotesUiDocument.CurrentField

- **DialogBoxCanceled**
  gibt an, ob eine Dialogbox mit der Abbruch-Taste verlassen wurde.
  True oder False = NotesUiDocument.DialogBoxCanceled

- **Document**
  liefert das Backend-Objekt der Klasse NotesDocument für das gerade geöffnete Dokument.
  Set NotesDocument = NotesUiDocument.Document

- **EditMode**
  kann dazu dienen, anzuzeigen, ob sich das aktuelle Dokument im „Bearbeiten"-Modus befindet. Außerdem kann damit auch der „Bearbeiten"-Modus ein- bzw. ausgeschaltet werden.
  True oder False = NotesUiDocument.EditMode
  NotesUiDocument.EditMode = True oder False

- **FieldHelp**
  steuert die Anzeige der Hilfezeile des Notes-Clients am unteren Rand des Dokumentenfensters.
  True oder False = NotesUiDocument.FieldHelp
  NotesUiDocument.FieldHelp = True oder False

- **HiddenChars**
  zeigt die internen Steuerzeichen (wie Zeilenschaltung oder Tabulatortaste) an bzw. nicht an.
  True oder False = NotesUiDocument.HiddenChars
  NotesUiDocument.HiddenChars = True oder False

- **HorzScrollBar**
  zeigt die horizontale Bildlaufleiste an bzw. nicht an.
  True oder False = NotesUiDocument.HorzScrollBar
  NotesUiDocument.HorzScrollBar = True oder False

- **InPreviewPane**
  gibt an, ob das aktuelle Dokument in dem Dokumentenvorschaufenster geöffnet ist.
  True oder False = NotesUiDocument.InPreViewPane

- **IsNewDoc**
  Diese Eigenschaft besagt, dass das aktuelle Dokument neu angelegt und noch nicht gesichert wurde.
  True oder False = NotesUiDocument.IsNewDoc

- **PreviewDocLink**
  gibt an oder legt fest, ob die Dokumentenvorschau angezeigt wird.
  True oder False = NotesUiDocument.PreViewDocLink
  NotesUiDocument.PreViewDocLink = True oder False

- **PreviewParentDoc**
  gibt an oder legt fest, ob das Elterndokument in der Dokumentenvorschau angezeigt wird.
  True oder False = NotesUiDocument.PreViewParentDoc
  NotesUiDocument.PreViewParentDoc = True oder False

- **Ruler**
  schaltet das Lineal am oberen Fensterrand ein bzw. aus.
  True oder False = NotesUiDocument.Ruler
  NotesUiDocument.Ruler = True oder False

- **WindowTitle**
  enthält den Titel des Dokumentenfensters.
  Name des Dokumentenfensters =
  NotesUiDocument.WindowTitle

Programmtext

```
Dim ws as New NotesUiWorkspace
Dim uidoc as NotesUiDocument
Dim doc as NotesDocument
Dim strFeldname as String

,/// Dokumentenobjekt erstellen
Set uidoc = ws.CurrentDocument

,/// Backend-Objekt ermitteln
Set doc = uidoc.Document

,/// Name des Feldes ermitteln, in dem der Cursor steht
strFeldname = uidoc.CurrentField
```

```
,/// Verschiedene Anzeige-Elemente des aktuellen Dokumentes
,/// einrichten:

,/// Automatische Aktualisierung
uidoc.AutoReload = True

,/// Bearbeitungsmodus einschalten
if not uidoc.EditMode then uidoc.EditMode = True

,/// Hilfezeile anzeigen
uidoc.FieldHelp = True

,/// Verborgene Zeichen ausschalten
uidoc.HiddenChars = False

,/// Sicherungshinweis bei einem neuen Dokument anzeigen
if uidoc.IsNewDoc then
    Msgbox „Sichern nicht vergessen, _
      MB_ICONEXCLAMATION, _
      „Achtung!“
End if

,/// Horizontale Bildlaufleiste anzeigen
uidoc.HorzScrollBar = True

,/// Fenstertitel festlegen
uidoc.WindowsTitle = „Neues Dokument“
```

Die Methoden der NotesUiDocument-Klasse erlauben die Arbeit mit dem geöffneten Dokument und seinen Feldern:

- **Categorize**
  ordnet ein Dokument einer Kategorie zu.
  Call NotesUiDocument.Categorize ([Kategorie])

- **Clear**
  löscht die markierten Zeichen und Objekte in einem Feld eines Dokumentes, ohne sie in die Zwischenablage zu kopieren.
  Call NotesUiDocument.Clear

- **Close**
  schließt das aktuelle Dokument und das dazugehörige Fenster.
  Call NotesUiDocument.Close

- **CollapseAllSections**
  schließt alle aufklappbaren Abschnitte eines Dokumentes.
  Call NotesUiDocument.CollapseAllSections

- **Copy**
  kopiert die markierten Zeichen und Objekte in die Zwischenablage von Windows.
  Call NotesUiDocument.Copy

- **CreateObject**
  erstellt ein OLE-Objekt in einem Rich-Text-Feld eines Dokumentes.
  Objekt = NotesUiDocument.CreateObject ([Name [, Art [, DateiPfad]]]

- **Cut**
  kopiert den markierten Feldinhalt in die Zwischenablage und löscht ihn.
  Call NotesUiDocument.Cut

- **DeleteDocument**
  markiert das aktuelle Dokument als gelöscht und schließt es.
  Call NotesUiDocument.DeleteDocument

- **DeselectAll**
  macht alle Markierungen im aktuellen Dokument rückgängig.
  Call NotesUiDocument.DeselectAll

- **ExpandAllSections**
  klappt alle Abschnitte im Dokument auf.
  Call NotesUiDocument.ExpandAllSections

- **FieldAppendText**
  hängt eine Zeichenkette an ein bestehendes Feld an.
  Call NotesUiDocument.FieldAppendText (Feldname, Text)

- **FieldClear**
  löscht den Inhalt eines Feldes.
  Call NotesUiDocument.FieldClear (Feldname)

- **FieldContains**
  überprüft ein Feld daraufhin, ob es einen bestimmten Text enthält.
  True oder False =
  NotesUiDocument.FieldContains (Feldname, Wert)

- **FieldGetText**
  übernimmt den Inhalt eines Feldes in eine Zeichenkette. Zahlen- oder Datumswerte werden in Zeichen konvertiert.
  Textvariable =
  NotesUiDocument.FieldGetText (Feldname)

- **FieldSetText**
  füllt ein Feld aus einer Zeichenkette. Ist das Feld mit Werten gefüllt, so werden diese überschrieben. Die Zeichen werden automatisch in den Typ des Feldes konvertiert.
  Call NotesUiDocument.FieldSetText (Feldname,
  Textvariable oder Zeichenkette)

- **FindFreeTimeDialog**, **FindFreeTimeDialogEx**
  zeigt das Dialogfenster für die „Freie-Zeit-Suche“ an und ermöglicht die Ermittlung einer gemeinsamen Besprechungszeit.
  True oder False = NotesUiDocument.FindFreeTimeDialog
  (Name des Feldes, in dem die Besprechungsteilnehmer gespeichert sind, Feldname für optionale Besprechungsteilnehmer, Feldname der gewünschten Besprechungsräume, Feldname mit den geplanten Ressourcen, Feldname mit den optionalen Ressourcen, Feldname für die Besprechungsteilnehmer, die an der Besprechung nicht teilnehmen sollen, Feldname, in der die Anfangszeit zu finden ist, Feldname für die Beendigungszeit)

- **FindString**
  sucht im angegebenen Feld nach dem genannten String.
  Call NotesUiDocument.FindString (Suchbegriff
  [, ganzes Wort suchen] [, rückwärts suchen]
  [, Groß-Kleinschreibung beachten]

[, Ersatzzeichen verwenden] [, Akzente beachten]
[, ganzes Dokument durchsuchen]
[, nächstes Vorkommen verwenden]
[, Umbruch beachten]
[, Trennzeichen ignorieren])

- **Forward**
  Dokument als e-Mail versenden.
  Call NotesUiDocument.Forward

- **GetObject**
  liefert ein Objekt für den angegebenen Objektnamen.
  Set Object =
  NotesUiDocument.GetObject (Name des Objektes)

- **GetSelectedText**
  kopiert den markierten Text in eine Textvariable.
  Textvariable =
  NotesUiDocument.GetSelectedText (Name des Feldes)

- **GotoBottom**
  setzt den Mauszeiger in das letzte editierbare Feld des Dokumentes.
  Call NotesUiDocument.GotoBottom

- **GotoField**
  positioniert den Cursor in das angegebene Feld des Dokumentes.
  Call NotesUiDocument.GotoField (Name des Feldes)

- **GotoNextField**
  positioniert den Cursor in das nächste Feld des Dokumentes. Die Suchrichtung verläuft dabei nach rechts und nach unten.
  Call NotesUiDocument.GotoNextField

- **GotoPrevField**
  positioniert den Cursor in das vorhergehende Feld des Dokumentes.
  Call NotesUiDocument.GotoPrevField

- **GotoTop**
  setzt den Mauszeiger in das erste editierbare Feld des Dokumentes.
  Call NotesUiDocument.GotoTop

- **Import**
  importiert eine Datei in das aktuelle Dokument.
  Call NotesUiDocument.Import (Filter, Datei-Name)

- **InsertText**
  fügt an der aktuellen Mauszeigerposition den angegebenen Text ein.
  Call NotesUiDocument.InsertT0ext (Textvariable oder Zeichenkette)

- **NavBarSetText, NavBarSpinnerStart, NavBarSpinnerStop**
  Verwendung der Adresseingabe des Internet Explorers.

- **Paste**
  fügt den Inhalt der Windows-Zwischenablage an der Cursorposition ein.
  Call NotesUiDocument.Paste

- **Print**
  druckt das aktuelle Dokument.
  Call NotesUiDocument.Print ([Anzahl Kopien [, Startseite [, Endseite [, Entwurfsqualität]]]])

- **Refresh**
  Dokument aktualisieren und Felder neu berechnen.
  Call NotesUiDocument.Refresh ([Rich-Text-Felder einbeziehen])

- **RefreshHideFormulas**
  berechnet alle Formeln neu, die bei der Eigenschaft „Verbergen wenn“ der Maskenobjekte hinterlegt sind. Die übrigen Maskenformeln werden nicht neu berechnet.
  Call NotesUiDocument.RefreshHideFormulas

- **Reload**
  aktualisiert die Anzeige und die Arbeitsspeicherinhalte des aktuellen Dokumentes aus dem gespeicherten Backend-Dokument.
  Call NotesUiDocument.Reload

- **Save**
  speichert das aktuelle Dokument.
  Call NotesUiDocument.Save

- **SaveNewVersion**
  speichert das aktuelle Dokument als neue Version.
  Call NotesUiDocument.SaveNewVersion

- **SelectAll**
  Ist das aktuelle Dokument im Bearbeitungsmodus, so wird der Inhalt des aktuellen Feldes markiert. Befindet sich das aktuelle Dokument im Lesemodus, wird das ganze Dokument markiert.
  Call NotesUiDocument.SelectAll

- **Send**
  versendet das aktuelle Dokument als e-Mail.
  Call NotesUiDocument.Send

- **SpellCheck**
  führt die Rechtschreibprüfung für das aktuelle Dokument aus.
  Call NotesUiDocument.SpellCheck

Programmtext

```
    ,/// Beispiel-Funktion für die Übernahme des aktuellen
    ,/// Dokumentes in ein Aufgaben-Dokument der Mail-Datenbank
    ,/// des Benutzers

Sub CreateToDo
,///
    ,/// Objekt für den Arbeitsbereich
    Dim ws As New NotesUIWorkspace

    ,/// Objekt für die aktuelle Sitzung
    Dim ses As New NotesSession

    ,/// Objekt für die Mail-Datenbank des Benutzers
    Dim dbMail As NotesDatabase

    ,/// Objekt für das Backend-Dokument des
    ,/// geöffneten Frontend-Dokuments
    Dim doc As NotesDocument

    ,/// Backend-Objekt für die Aufgabe
    Dim docToDo As NotesDocument

    ,/// Frontend-Objekt des aktuell geöffneten Dokumentes
    Dim uidoc As NotesUIDocument
```

```
,/// Frontend-Objekt für die Aufgabe
Dim uidocToDo As NotesUIDocument

,/// NotesItem-Objekt für das Kopieren der Felder
Dim item As NotesItem

,/// Mail-Datenbank des Benutzers
Dim strUserMailDb As String

,/// Backend-Object für die aktuelle Datenbank
Set db = ses.CurrentDatabase

,/// Objekt für das aktuelle Dokument
Set uidoc = ws.CurrentDocument

,/// Mail-Datei des Benutzers aus der Datei
,/// „Notes.ini“ ermitteln
strUserMailDb = _
  CStr (ses.GetEnvironmentString („MailFile“, True)

,/// geöffnetes Dokument sichern
Call uidoc.Save

,/// als Backend-Objekt öffnen
Set doc = uidoc.Document

,/// Objekt für die Mail-Datenbank
Set dbMail = New NotesDatabase (db.Server, _
  strUserMailDb)

,/// Backend-Objekt für das neue Dokument
Set docToDo = dbMail.CreateDocument
```

```
    ,/// Maske zuordnen
    docToDo.Form = „Task“

    ,/// Dokumentfelder füllen
    docToDo.Subject = uidoc.FieldGetText („Betreff“)
    docToDo.Importance = „2“
    docToDo.chair = ses.UserName
    docToDo.AssignState = 0
    Set item = doc.GetFirstItem („datRemind“)
    Call docToDo.CopyItem (item, „StartDate“)
    Set item = doc.GetFirstItem („datEnd“)
    Call docToDo.CopyItem (item, „DueDate“)
    Set item = doc.GetFirstItem („Body“)
    Call docToDo.CopyItem (item, „Body“)

    ,/// Das neue Dokument im Bearbeitungsmodus anzeigen
    Call ws.EditDocument (True, docToDo)
,///
End Sub
```

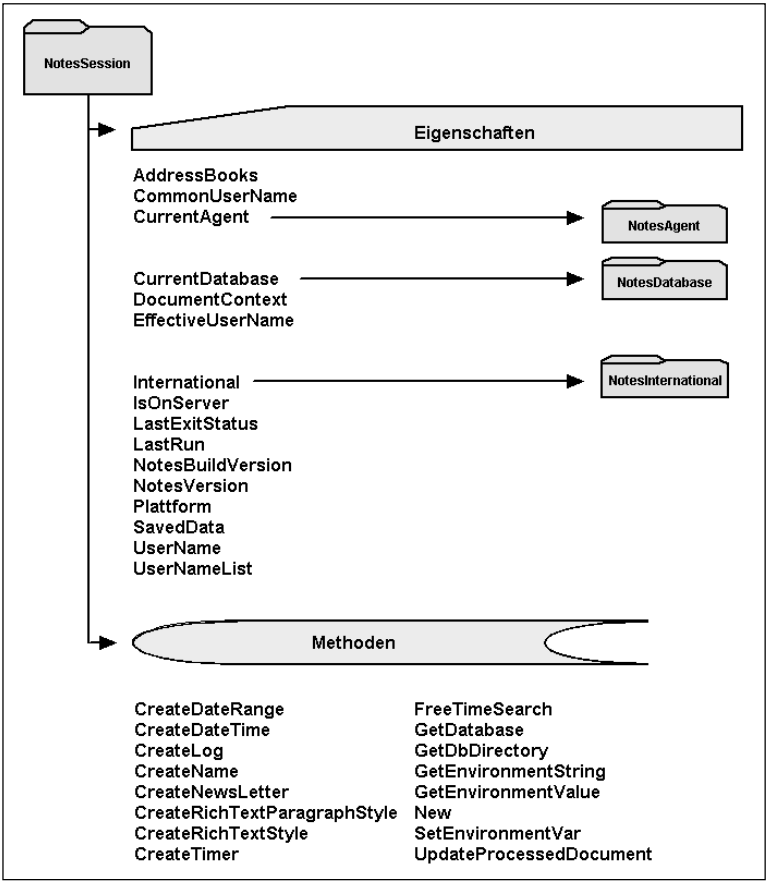
Weitere Beispiele für den Einsatz der Frontend-Klassen sind im Kapitel „Anwendungsbeispiele“ zu finden.

6.3.6.2 Backend-Klassen

Notes-Backend-Klassen enthalten den größten Teil des Notes- / Domino-Objekt-Modells. Hier stehen die Werkzeuge zur Verfügung, die auf beliebige Notes-Datenbanken und deren Elemente und sogar auf externe Daten zugreifen. Ähnlich wie die Frontend-Klassen steht auch bei den Backend-Klassen eine Ausgangsklasse zur Verfügung, die wichtige Daten für den Zugriff auf andere Bakkend-Klassen bereitstellt. Es handelt sich hier um die Klasse NotesSession.

6.3.6.2.1 NotesSession

Das NotesSession-Objekt bildet die Basis für die Backend-Klassen. Von diesem Objekt aus kann auf die verschiedenen Objekte zugegriffen werden, die von dem Festspeicher geladen werden sollen. Neben der gerade geöffneten Datenbank erlaubt das NotesSession-Objekt den Zugriff auf die Umgebungsvariable, den aktuellen Benutzer, Dokumentsammlungen sowie auf die verwendete Notes-Version. Da die Eigenschaften der Sitzung immer auf dem neuesten Stand sind, kann das NotesSession-Objekt immer wieder neu angelegt werden, wenn es auch bereits als Objekt besteht.



Die Klasse NotesSession

Ein Objekt der NotesSession-Klasse wird mit der Methode “New” erstellt. Das erfolgt in der Regel direkt bei seiner Definition:

Dim session as New NotesSession

Dann wird mit den Eigenschaften und Methoden dieses Objektes weitergearbeitet. Folgende Eigenschaften des NotesSession-Objektes stehen damit zur Verfügung:

- **Addressbooks**
  Ein Array von Datenbanken, die in dem aktuellen Umfeld als Adressbücher zur Verfügung stehen. Dabei kann es sich sowohl um öffentliche Adressbücher, als auch um das private Adressbuch des Benutzers handeln.
  Variant mit Datenbankobjekten = _ notesSession.Addressbooks

Programmtext

```
,/// NotesSession-Objekt erstellen
Dim ses as New NotesSession

,/// Variant-Variable für die Datenbank-Objekte
Dim varBooks as Variant

,/// Ansicht „People“ aus der Adressdatenbank
Dim vwPeople as NotesView

,/// Dokument des Benutzers
Dim doc as NotesDocument

,/// Adressdatenbank-Objekte erstellen
varBooks = ses.Addressbooks
```

```
,/// In allen Adressdatenbanken nach dem Namen des aktuellen
,/// Benutzers suchen
Forall b in varBooks
    ,/// Datenbank öffnen
    Call b.open(„“, „“)

    ,/// Ansichts-Objekt erstellen
    Set vwPeople = b.GetView(„People“)

    ,/// Dokument-Objekt erstellen
    Set doc = _
    vwPeople.GetDocumentByKey(ses.CommonUserName)

    ,/// In der Statuszeile das Suchergebnis anzeigen
    If Not doc is Nothing Then
      print b.Title; „ enthält „; doc.FullName
    else
      print „In „; b.Title; „ „; ses.CommonUserName; _
            „ nicht gefunden!“
    End If

    End Forall
```

- **CommonUserName**
  Diese Eigenschaft enthält den Vor- und Nachnamen des aktuellen Benutzers.
  strUser = NotesSession.CommonUserName

Programmtext

```
Dim ses as New NotesSession
Dim strUser as String

,/// Name in der Form „Peter Muster“ aus der
,/// aktuellen Sitzung holen
strUser = ses.CommonUserName

,/// Ergebnis in der Statuszeile anzeigen
Print „Hallo, „; strUser; „ wie geht es heute?“
```

- **CurrentAgent**
  Liefert den gerade aktiven Agenten als Objekt.
  Set NotesAgent = NotesSession.CurrentAgent
- **CurrentDatabase**
  Objekt der Datenbank, in der das LotusScript-Programm hinterlegt ist.
  Set NotesDatabase = NotesSession.CurrentDatabase

Das Datenbank-Objekt kann auch direkt als Unterobjekt verwendet werden:

strTitel = session.CurrentDatabase.Title

Programmtext

```
Dim ses as New NotesSession
Dim db as NotesDatabase
Dim strTitel as String

Set db = ses.CurrentDatabase
strTitel = db.Title

,/// oder
print ses.CurrentDatabase.Title
```

- **DocumentContent**
  Wenn ein externes Programm über die Notes-API ein Dokument-Objekt im Arbeitsspeicher erzeugt hat, kann mit dieser Eigenschaft darauf zugegriffen werden.
  Für Web-Agenten liefert DocumentContent den Inhalt der CGI-Variablen.
  NotesDocument = NotesSession.DocumentContent

- **EffectiveUserName**
  Name des Benutzers, unter dem das aktuelle Programm läuft. Wird das Programm auf einer Arbeitsstation ausgeführt, erhält man den Namen des aktuellen Benutzers. Bei einer Ausführung auf dem Server, wird der Name des Benutzers zurückgeliefert, der das Programm zuletzt gespeichert (umgewandelt) hat.
  Name als String = NotesSession.EffectiveUserName

- **IsOnServer**
  Diese Eigenschaft gibt an, ob das aktuelle Programm auf dem Server ausgeführt wird. Mit Hilfe dieser Eigenschaft kann z.B. entschieden werden, ob auf das öffentliche oder das private Adressbuch zugegriffen wird.
  longVariable = NotesSession.IsOnServer

- **LastExitStatus**
  Fehlercode, den der Agent-Manager setzt, wenn ein Agent beendet wird. Das Ergebnis lautet 0, wenn ein Agent fehlerfrei beendet wird.
  intVariable = NotesSession.LastExitStatus

- **LastRun**
  Datum, an dem der aktuelle Agent zum letzten Mal ausgeführt wurde.
  varDatum = NotesSession.LastRun

- **NotesBuildVersion, NotesVersion**
  Diese Eigenschaft enthält die genaue Release-Version der ausführenden Software. Handelt es sich um eine Ausführung auf dem Server, so wird die Version der Server-Software übergeben. Bei der Ausführung auf einer Arbeitsstation wird die Version der Notes-Client-Software übergeben.
  longVariable = NotesSession.NotesBuildVersion
  stringVariable = NotesSession.NotesVersion

- **Platform**
  Name des Betriebssystems, unter dem das aktuelle Programm ausgeführt wird (z.B. Windows/32 oder UNIX)
  stringVariable = NotesSession.Platform

- **SavedData**
  Beim Speichern eines Agenten wird ein einzelnes Dokument erstellt, das jeweils zusammen mit dem Agenten verfügbar ist. Es enthält keine Felder. Wenn Informationen in diesem Dokument gespeichert werden, so stehen sie bei der nächsten Ausführung des Agenten wieder zur Verfügung. Nur im Rahmen von Agenten kann auf das dazu gehörige Dokument zugegriffen werden.
  Set doc = NotesSession.SavedData

- **UserName**
  Name des aktuellen Benutzers in hierarchischer Form. Wird das Programm auf dem Server ausgeführt, liefert UserName den Namen des Servers, wird es auf einer Arbeitsstation ausgeführt, liefert UserName den Namen des Benutzers.
  stringVariable = NotesSession.UserName

- **UserNameList**
  Array mit Namen des aktuellen Benutzers inklusive dem alternativen Benutzernamen.
  stringArray = NotesSession.UserNameList

Neben den Eigenschaften, die im Wesentlichen nur abrufbare Informationen enthalten, stellt die NotesSession-Klasse viele Methoden zur Verfügung, mit deren Hilfe Objekte erstellt und Werte bearbeitet werden können:

- **CreateDateRange**
  Erstellt ein Objekt der Klasse NotesDateRange mit Start- und Enddatum bzw. -zeit.
  Set NotesDateRange = NotesSession.CreateDateRange

- **CreateDateTime**
  Erstellt ein NotesDateTime-Objekt aus einem entsprechenden Datums-/Zeitstring.
  Set NotesDateTime = _
  NotesSession.CreateDateTime(stringVariable)

- **CreateLog**
  Erstellt ein Objekt der NotesLog-Klasse.
  Set NotesLog = NotesSession.CreateLog(stringVariable)

- **CreateName**
  Erstellt ein Objekt der Klasse NotesName aus einem Namensstring.
  Set NotesName = _
  NotesSession.CreateName(stringVariable, [Sprachcode])

- **CreateNewsletter**
  Erzeugt ein Objekt der Newsletter-Klasse aus einer Dokumentensammlung.
  Set NotesNewsletter = _
  NotesSession.CreateNewsletter(_
  NotesDocumentCollection)

- **CreateRichTextParagraphStyle**
  Diese Methode erzeugt ein Objekt für die Gestaltung eines Absatzes in einem Richtext-Feld.
  Set NotesRichTextParagraphStyle = _
  NotesSession.CreateRichTextParagraphStyle

- **CreateTimer**
  Ein Timer-Objekt dient dazu, ein Programm in einem bestimmten Zyklus auszuführen. Die Dauer und das Intervall werden als Eigenschaften des Timer-Objektes festgelegt.
  NotesTimer = NotesSession.CreateTimer

- **FreeTimeSearch**
  Durchsucht Kalenderdatenbanken nach freier Zeit für alle Teilnehmer einer Besprechung.
  Set NotesDateRange = _
  NotesSession.FreeTimeSearch(NotesDateRange, _
  intDauer, stringArray, intErsteGefundene)

- **GetDatabase**
  Erstellt ein neues Datenbank-Objekt und öffnet, wenn möglich, die angegebene Datenbank.
  NotesDatabase = NotesSession.GetDatabase(_
  stringServer, stringDatenbank, intAnlegen)

Programmtext

```
,/// Objekt für die Session erzeugen
Dim ses as New NotesSession
Dim db as NotesDatabase
Dim vw as NotesView

,/// Datenbankobjekt erzeugen
Set db = ses.GetDatabase(ses.Server, „mail\wschulte.nsf“, _
    FALSE)
,/// statt ses.Server könnte auch der Name eines Servers stehen
,/// also z.B. „miri/wse-edv/de“

,/// Objekt für die Kalender-Ansicht erzeugen
Set vw = db.GetView(„$Calendar“)
,/// oder mit Hilfe der New-Methode des
,/// NotesDatabase-Objektes:
Set db = new NotesDatabase(ses.Server, „mail\wschulte.nsf“)
```

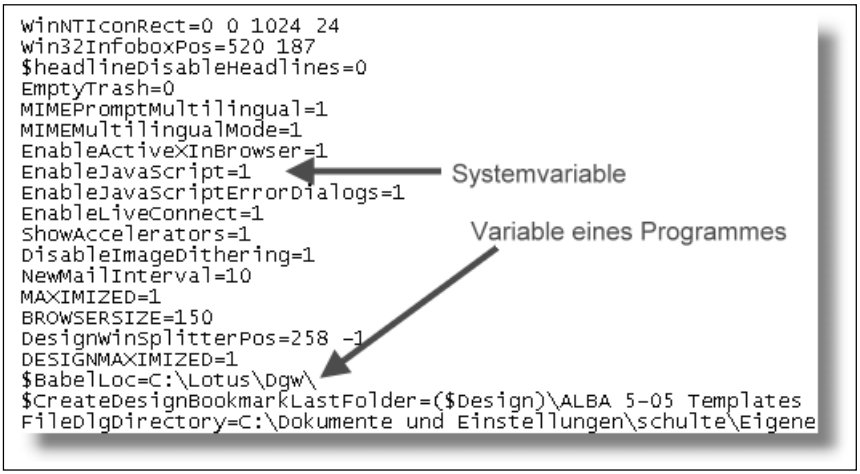
- **GetDbDirectory**
  Erzeugt ein Objekt der Klasse NotesDbDirectory für die Datenbanken auf dem angegebenen Server.
  NotesDbDirectory = NotesSession.GetDbDirectory(_
  stringServer)

Programmtext

```
Dim ses as New NotesSession
Dim dirDb as NotesDbDirectory
Dim db as NotesDatabase
Dim lngBytes as Long

,/// Weitermachen, wenn eine Datenbank nicht
,/// verarbeitet werden kann
On Error Resume Next

,/// Directory-Objekt erstellen
Set dirDb = ses.GetDbDirectory( _
    ses.CurrentDatabase.Server)

,/// Erste Datenbank
Set db = dirDb.GetFirstDatabase

While Not db Is Nothing
    ,/// Datenbank öffnen
    db.Open „“, „“

    ,/// Ermitteln, ob weniger als 90 % der Datenbank
    ,/// benutzbar sind
```

```
    If db.PercentUsed < 90 then
      ,/// Titel in der Statuszeile anzeigen
      Print db.Title; „ -> wird komprimiert“

      ,/// Datenbank komprimieren und
      ,/// freigegebenen Speicherplatz aufaddieren
      lngBytes = lngBytes + db.Compact
    End If

    ,/// Nächste Datenbank
    Set db = dirDb.GetNextDatabase
  Wend

  MsgBox lngBytes & „ Speicherplatz (in KB) freigegeben!“, _
    MB_ICONEXCLAMATION, „Achtung!“
```

- **GetEnvironmentString**
  Diese Methode holt den Wert einer bestimmten Variablen aus der Datei “Notes.ini” als Variant-Variable. Da Notes die Werte der Datei “Notes.ini” nur beim Starten des Programms lädt und dann aus dem Zwischenspeicher arbeitet, ist der Wert allerdings nicht unbedingt auf dem neuesten Stand.
  stringVariable = NotesSession.GetEnvironmentString(_ stringName[, intSystem])
  Dabei bedeutet “intSystem = TRUE”, dass kein “$” vor den Namen der Variablen gesetzt wird. Die Variablen ohne $-Zeichen sollten Lotus als Systemvariable vorbehalten bleiben.



Variable in der Notes-Ini.tif

- **GetEnvironmentValue**
  Im Gegensatz zu GetEvironmentString ist das Ergebnis dieser Methode eine Variant-Variable mit einer Zahl als Element.
  intVariable = NetesSession.GetEnvironmentValue(_ stringName [, intSystem])

- **SetEnvironmentVar**
  Erstellt oder ändert einen Eintrag in der Datei “Notes.ini”.
  Call NotesSession.SetEnvironmentVar(stringName, _ variantWert [, intSystem])

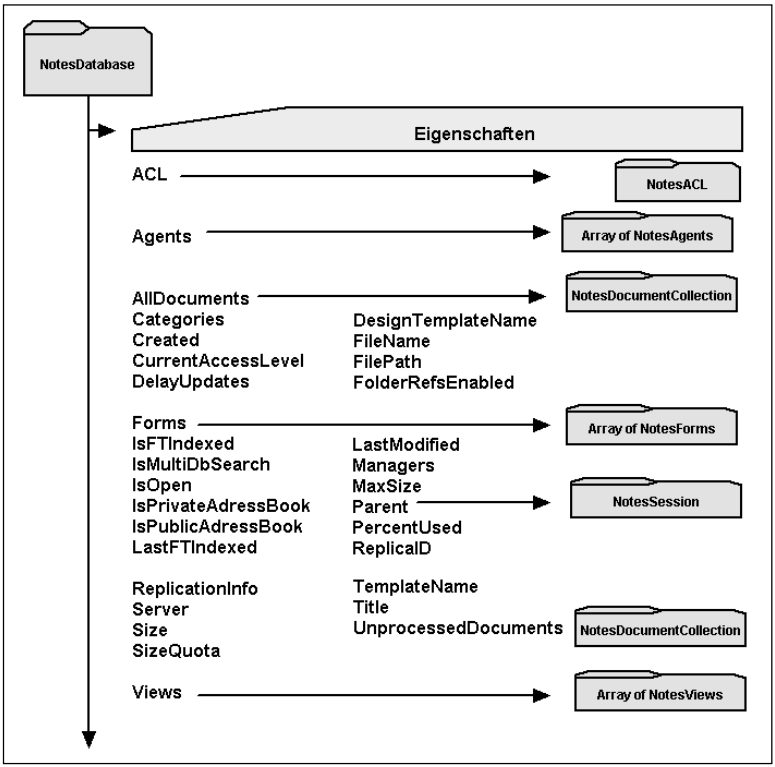
Programmtext

```
,/// Session-Objekt erstellen
Dim ses as New NotesSession
Dim strDir as String

,/// Eintrag „FileDlgDirectory“ aus der Datei „Notes.ini“
,/// übernehmen und am Bildschirm ausgeben
strDir = ses.GetEnvironmentString(„FileDlgDirectory“, True)
msgbox strDir, 48, „FileDlgDirectory“

,/// Eintrag „FileDlgDirectory“ in der Datei „Notes.ini“
,/// ändern
ses.SetEnvironmentVar „FileDlgDirectory“, „C:\“, True

,/// Eintrag „FileDlgDirectory“ aus der Datei „Notes.ini“
,/// übernehmen und am Bildschirm ausgeben
strDir = ses.GetEnvironmentString(„FileDlgDirectory“, True)
msgbox strDir, 48, „FileDlgDirectory zum Zweiten“
```

- **UpdateProcessedDoc**
  Kennzeichnet ein Dokument als “Bearbeitet” durch einen Agenten. Die Kennzeichnung wirkt nur innerhalb eines Agenten.
  Call NotesSession.UpdateProcessedDoc(NotesDocument)

6.3.6.2.2 Die Klasse NotesDatabase

Notes-Datenbanken sind die Container für Daten und Programme. Sehr viele Programme greifen deshalb auf Datenbanken zu. Das Datenbankobjekt kann über das NotesSession-Objekt (z.B. die aktuelle Datenbank und die GetDatabase-Methode) oder über die New-Methode des NotesDatabase-Objektes erstellt werden.



Eigenschaften des NotesDatabase-Objektes

Entsprechend der Bedeutung des Database-Objektes werden auch die Eigenschaften der Datenbank benötigt. Folgende Eigenschaften stehen zur Verfügung:

- **ACL**
  Diese Eigenschaft liefert den Inhalt der Zugriffskontrollliste einer Datenbank als NotesACL-Objekt.
  Set NotesACL = NotesDatabase.ACL

- **Agents**
  liefert die in einer Datenbank vorhandenen Agenten in Form einer Array von NotesAgent-Objekten.
  arrayNotesAgents = NotesDatabase.Agents

- **AllDocuments**
  Diese Methode erstellt eine Sammlung aller Dokumente der Datenbank als NotesDocumentCollection.
  Set NotesDocumentCollection = _
  NotesDatabase.AllDocuments

Programmtext

```
Dim ses as New NotesSession
Dim db as NotesDatabase
Dim col as NotesDocumentCollection
Dim doc as NotesDocument

,/// Datenbank-Objekt für die aktuelle Datenbank erzeugen
Set db = ses.CurrentDatabase

,/// Dokumentsammlung aus allen Dokumenten der Datenbank
,/// erstellen
Set col = db.AllDocuments
```

```
,/// Wenn die Datenbank Dokumente enthält
If col.count > 0 Then

   For i = 1 to col.count
      ,/// Dokument-Objekt erzeugen
      Set doc = col.GetNthDocument(i)

      ,///Betreff-Feld in der Statuszeile anzeigen
      Print doc.Subject(0)
   Next i

End If
```

- **Categories**
  Suchbegriffe, unter denen die Datenbank im Datenbank-Katalog aufgeführt wird.
  variantString = NotesDatabase.Categories
  NotesDatabase.Categories = variantString

- **Created**
  liefert das Datum, an dem die Datenbank erstellt wurde.
  variantDatum = NotesDatabase.Created

- **CurrentAccessLevel**
  Diese Eigenschaft enthält die Zugriffsrechte des aktuellen Benutzers. Auf einer Arbeitsstation sind das die Rechte des Benutzers, der gerade angemeldet ist, auf einem Server gelten die Rechte des Benutzers, der das Programm zuletzt gesichert hat.
  intKonstante = NotesDatabase.CurrentLevelAccess

Folgende Konstante sind verfügbar:

ACLLEVEL_NOACCESS	Kein Zugriff
ACLLEVEL_DEPOSITOR	Archivar
ACLLEVEL_READER	Leser
ACLLEVEL_AUTHOR	Autor
ACLLEVEL_EDITOR	Editor
ACLLEVEL_DESIGNER	Designer
ACLLEVEL_MANAGER	Manager

- **DelayUpdates**
  gibt an, ob die Datenbankzugriffe auf dem Server verzögert erfolgen, um die Performance zu verbessern.
  intVariable = NotesDatabase.DelayUpdates
  NotesDatabase.DelayUpdates = intVariable

- **DesignTemplateName**
  Name der Datenbankschablone, aus der die Datenbank ihre Gestaltung bezieht.
  stringVariable = NotesDatabase.DesignTemplateName

- **FileName**
  Dateiname der Datenbank einschließlich der Datei-Erweiterung (z.B. „.nsf“), aber ohne das Windowsverzeichnis.
  stringVariable = NotesDatabase.FileName

- **FilePath**
  Name des Windowsverzeichnisses und der Datei (z.B. „mail\wschulte.nsf“). Auf einer Arbeitsstation wird der komplette Pfad (z.B. „C:\Lotus\Notes\Data\Mail\wschulte.nsf“) ermittelt. Auf dem Server liefert FilePath die Verzeichnisse unterhalb des Data-Verzeichnisses.
  stringVariable = NotesDatabase.FilePath

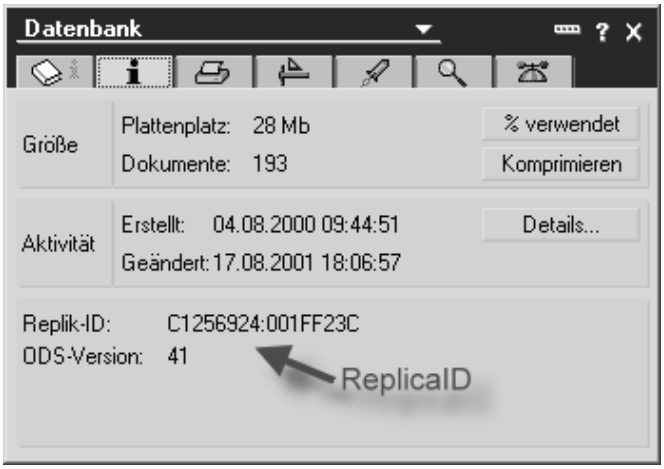
Programmtext

```
Dim ses as New NotesSession
Dim strFileName as String
Dim strFilePath as String

,/// Dateiname ermitteln
strFileName = ses.CurrentDatabase.FileName

,/// Verzeichnis einschließlich Dateiname
strFilePath = ses.CurrentDatabase.FilePath

,/// Verzeichnis anzeigen
MsgBox „Das Verzeichnis der Datenbank „ & chr(10) & _
    ses.CurrentDatabase.Title & chr(10) & _
    „ heißt: „ & chr(10) & chr(10) & _
    strLeft(strFilePath, strFileName), 48, „Information!“
```

- **FolderReferencesEnabled**
  Zeigt an, ob verwaltet wird, welche Dokumente in welchen Ordnern abgelegt sind.
  intVariable = NotesDatabase.FolderReferencesEnabled
  NotesDatabase.FolderReferencesEnabled = intVariable
- **Forms**
  enthält alle Masken einer Datenbank als Array von NotesForm-Objekten.
  arrayForms = NotesDatabase.Forms

- **IsDirectoryCatalog**
  Diese Eigenschaft gibt an, ob die Datenbank ein Adressverzeichnis ist.
  intVariable = NotesDatabase.IsDirectoryCatalog
- **IsFTIndexed**
  Wird für die Datenbank ein Volltext-Index geführt, so ist diese Eigenschaft auf TRUE gesetzt.
  intVariable = NotesDatabase.IsFTIndexed
- **IsMultiDbSearch**
  zeigt an, ob die Datenbank einen Mehrfachdatenbank-Index repräsentiert.
  intVariable = NotesDatabase.IsMultiDbSearch
- **IsOpen**
  zeigt an, ob eine Datenbank geöffnet ist. Damit kann z.B. getestet werden, ob die Datenbank auf einem bestimmten Server existiert.
  intVariable = NotesDatabase.IsOpen
- **IsPrivateAddressBook, IsPublicAddressBook**
  gibt aus, ob die aktuelle Datenbank ein privates oder öffentliches Adressverzeichnis ist.
  intVariable = NotesDatabase.IsPublicAddressBook
- **LastFTIndexed**
  enthält das Datum, an dem der Volltext-Index der Datenbank zuletzt aktualisiert wurde.
  variantDatum = NotesDatabase.LastFTIndexed
- **Managers**
  zeigt die Personen, Server und Gruppen an, die Managerrechte an der Datenbank haben.
  arrayStrings = NotesDatabase.Managers

- **MaxSize**
  zeigt die festgelegte Maximalgröße der Datenbank an. Standardmäßig ist die Größe ab Release 5 unbegrenzt.
  longVariable = NotesDatabase.MaxSize
- **Parent**
  enthält das NotesSession-Objekt, zu dem die Datenbank gehört.
  NotesSession = NotesDatabase.Parent
- **PercentUsed**
  Um den nicht mehr benutzten Anteil der Datenbank zu erhalten, wird der Inhalt dieser Eigenschaft angezeigt. Ist der Prozentsatz hoch, sollte die Datenbank komprimiert werden.
  doubleVariable = NotesDatabase.PercentUsed
- **ReplicaID**
  Notes kennzeichnet alle Repliken einer Datenbank mit einer einheitlichen Kennung.
  stringVariable = NotesDatabase.ReplicaID



ReplicaID

- **ReplicationInfo**
  liefert das Objekt NotesReplication zu der aktuellen Datenbank.
  NotesReplication = NotesDatabase.ReplicationInfo

- **Server**
  Name des Servers, auf dem die aktuelle Datenbank gespeichert ist. Ist die Datenbank auf einer Arbeitsstation gespeichert, wird ein leerer String („“) ausgegeben.
  stringVariable = NotesDatabase.Server

- **Size**
  enthält die Größe der Datenbank in Bytes.
  doubleVariable = NotesDatabase.Size

- **SizeQuota**
  Vorgegebener Speicherplatz, der maximal für die Datenbank auf dem Server zur Verfügung stehen soll.
  longVariable = NotesDatabase.SizeQuota
  NotesDatabase.SizeQuota = longVariable

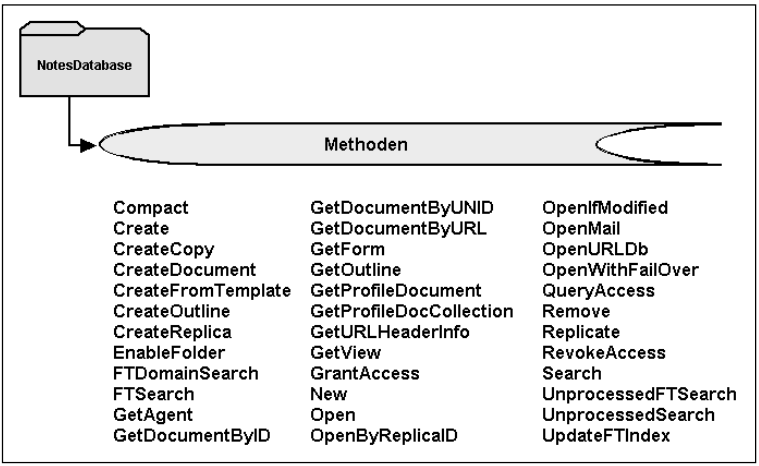
- **TemplateName**
  Name der Datenbankschablone, unter dem die Datenbank als Gestaltungsvorlage verfügbar ist.
  stringVariable = NotesDatabase.TemplateName

- **Title**
  Titel der Datenbank, der unter Notes angezeigt wird.
  stringVariable = NotesDatabase.Title
  NotesDatabase.Title = stringVariable
  UnprocessedDocuments
  alle Dokumente, die in der aktuellen Situation als “unverarbeitet” gelten. In einer Ansicht sind das alle markierten Dokumente.
  NotesDocumentCollection = _
  NotesDatabase.UnprocessedDocuments

Programmtext

```
Dim ses as New NotesSession
Dim col as NotesDocumentCollection
Dim doc as NotesDocument

,/// Dokumentsammlung erstellen
Set col = ses.CurrentDatabase.UnprocessedDocuments

,/// Prüfen, ob Dokumente markiert sind
If col.count > 0 Then
    Set doc = col.GetFirstDocument

    While Not doc Is Nothing
      Print doc.Subject(0)
      Set doc = col.GetNextDocument(doc)
    Wend
Else
    ,/// Fehlermeldung
    Msgbox _
    „Sie müssen mindestens 1 Dokument markieren!“, _
    MB_ICONSTOP, „Achtung!“
End If
```

- **Views**
  Alle Ansichten der Datenbank sind als Objekt in dieser Eigenschaft enthalten.
  arrayNotesViews = NotesDatabase.Views
  Die Methoden des Datenbankobjektes beziehen sich im Wesentlichen auf die Datenbank selbst und auf die Dokument-Objekte in der Datenbank.



Methoden des NotesDatabase-Objektes

- **Compact**
  Diese Methode erlaubt es, eine lokale Datenbank zu komprimieren. Dabei kann sie sich nicht auf die aktuelle Datenbank beziehen. Das Ergebnis der Methode ist die Differenz zwischen der Größe der Datenbank vor der Komprimierung und der Größe danach als Long-Integerzahl.
  longVariable = NotesDatabase.Compact

- **Create**
  erstellt eine Datenbank auf dem Festplattenspeicher. Die maximale Größe kann nur bei Datenbanken für Release 4 angegeben werden und höchstens 4 (GigaByte) betragen.
  Call NotesDatabase.Create(stringServer, stringDatei, _ intÖffnen[, intMaxGroesse])

- **CreateCopy**
  Diese Methode erzeugt eine leere Kopie der aktuellen Datenbank mit den gleichen Gestaltungselementen, der gleichen Zugriffskontrollliste und dem gleichen Titel. Die neue Datenbank ist keine Replik der aktuellen Datenbank und enthält keine Dokumente.
  Set NotesDatabase = NotesDatabase.CreateCopy(_ stringServer, stringDatei[, intMaxGroesse])

- **CreateDocument**
  In der angegebenen Datenbank wird ein neues Dokument als Objekt erzeugt.
  Set NotesDocument = NotesDatabase.CreateDocument

- **CreateFromTemplate**
  erstellt eine neue Datenbank aus der angegebenen Gestaltungsvorlage. Es kann festgelegt werden, ob zukünftige Gestaltungsänderungen aus der Vorlage übernommen werden sollen.
  Set NotesDatabase = _ NotesDatabase.CreateFromTamplate(stringServer, _ stringDatei, intÄnderungen[, intMaxGroesse])

- **CreateOutline**
  ermöglicht die Erstellung eines neuen Gliederungsobjektes. Es kann bestimmt werden, ob Standardeinträge angelegt werden sollen.
  Set NotesOutline = NotesDatabase.CreateOutline(_
  stringGliederung[, intStandard])

- **CreateReplica**
  erzeugt eine Replik der aktuellen Datenbank auf einem anderen Server oder der lokalen Arbeitsstation.
  Set NotesDatabase = NotesDatabase.CreateReplica(_
  stringServer, stringDatei)

- **EnableFolder**
  Diese Methode dient zur Überprüfung, ob ein bestimmter Ordner existiert, bevor mit anderen Methoden auf den Ordner zugegriffen wird.
  Call NotesDatabase.EnableFolder(stringOrdner)

- **FTDomainSearch**
  durchsucht den Domänenkatalog nach bestimmten Suchbegriffen, die in einer Abfrage definiert sind.
  Set NotesDocument = NotesDatabase.FTDomainSearch(_
  stringQuery, intMaxDokumente[, intSortierung[, _
  intOptionen[, intStart[, intAnzahl[, stringMaske]]]]])

- **FTSearchMethode**
  führt eine Volltextsuche über alle Dokumente einer Datenbank aus.
  Set NotesDocumentCollection = _
  NotesDatabase.FTSearch(stringAbfrage, _
  intMaxDokumente[, intSortierung[, intOpitionen]])

- **GetAgent**
  Diese Methode erstellt ein NotesAgent-Objekt unter Verwendung des Agentennamens.
  Set NotesAgent = _
  NotesDatabase.GetAgent(stringVariable)

- **GetDocumentByID, GetDocumentByUNID**
  Hier wird die Note-ID des Dokumentes verwendet, um ein entsprechendes Dokumenten-Objekt zu erstellen. Das setzt voraus, dass die Note-ID des Dokumentes bekannt ist.
  Set NotesDocument = _
  NotesDatabase.GetDocumentByID(stringID)

- **GetDocumentByURL**
  erstellt ein Dokument-Objekt für die Web-Navigator-Datenbank.
  Set NotesDocument =
  NotesDatabase.GetDocumentByURL(stringURL[, reload[, urllist]
  [, stringzeichensatz][, stringWebUser][, stringKennwort][, stringBenutzerProxy]
  [, stringKennwortProxy][, intReturn])

- **GetForm**
  erstelllt ein NotesForm-Objekt nach dem angegebenen Namen.
  Set NotesForm = NotesDatabase(stringMaske)

- **GetOutline**
  erzeugt ein NotesOutline-Objekt mit dem angegebenen Namen einer Gliederung.
  Set NotesOutline = _
  NotesDatabase.GetOutline(stringGliederung)

- **GetProfileDocument**
  öffnet oder erzeugt ein Profildokument. Wichtig ist, dass Profildokumente nicht mit einem "Create"-Befehl oder einer "New"-Methode, sondern über die Methode "GetProfileDocument" erstellt werden.
  Set NotesDocument = NotesDatabase.GetProfileDocument(stringProfil[, stringBenutzer])

- **GetProfileDocumentCollection**
  liefert eine Sammlung von NotesDocument-Objekten, die zu einem Profil gehören.
  Set NotesDocumentCollection = _
  GetProfileDocumentCollection(stringProfil)

- **GetURLHeaderInfo**
  erzeugt eine Zeichenkette mit den Header-Informationen zu einer angegebenen Web-Adresse.
  stringHeader = NotesDatabase.GetURLHeaderInfo(_
  stringURL, stringName[, stringWeb-Benutzer] _
  [, stringKennwort] [, stringProxy-Benutzer] _
  [, stringProxyKennwort])

- **GetView**
  erstellt ein NotesView-Objekt anhand eines Namens für die Ansicht.
  NotesView = NotesDatabase.GetView(stringAnsicht)

Programmtext

```
,/// NotesSession-Objekt erzeugen
Dim ses as New NotesSession
Dim db as NotesDatabase
Dim vw as NotesView
Dim strDbName as string

,/// Dateiname der Datenbank
strDbName = „Test\Muster.nsf“

,/// Server der aktuellen Sitzung verwenden
Set db = New NotesDatabase(ses.Server, strDbName)

,/// Ansichtsobjekt erstellen
Set vw = db.GetView(„vwNameOrt“)

,/// Jetzt kann z.B. ein Dokument-Objekt aus der Ansicht
,/// angelegt werden.
```

- **GrantAccess**
  Diese Methode dient dazu, einem bestimmten Benutzer ein bestimmtes Zugriffsrecht zuzuweisen. Das setzt allerdings voraus, dass der Benutzer, unter dessen Namen das Programm ausgeführt wird, über ausreichende Rechte verfügt.
  Call NotesDatabase.GrantAccess(stringBenutzer, intArt)

Für "intArt" können folgende Konstanten verwendet werden:

ACLLEVEL_NOACCESS	Kein Zugriff
ACLLEVEL_DEPOSITOR	Archivar
ACLLEVEL_READER	Leser
ACLLEVEL_AUTHOR	Autor
ACLLEVEL_EDITOR	Editor
ACLLEVEL_DESIGNER	Entwickler
ACLLEVEL_MANAGER	Manager

- **Open**
  öffnet eine Datenbank und legt – falls noch nicht vorhanden – ein NotesDatabase-Objekt an.
  intErgebnis = NotesDatabase.Open(stringServer, _
  stringDatei)
  Wenn bereits ein Objekt für die Datenbank besteht, werden die Parameter als Leerstring übergeben (z.B. db.Open(„“, „“).

- **OpenByReplicaID**
  öffnet eine Datenbank und erstellt ein NotesDatabase-Objekt unter Verwendung der Replik-ID der Datenbank.
  intErgebnis = NotesDatabase.OpenBy ReplicaID(_stringServer, stringID)

- **OpenIfModified**
  öffnet eine Datenbank und erstellt ein NotesDatabase-Objekt, wenn die Datenbank seit einem bestimmten Datum geändert wurde.
  intErgebnis = NotesDatabase.OpenIfModified(_ stringServer, stringDatei, NotesDateTime)

- **OpenMail**
  erstellt ein NotesDatabase-Objekt für die Mail-Datenbank des aktuellen Benutzers und öffnet die Datenbank.
  Call NotesDatabase.OpenMail

- **OpenURLDb**
  öffnet die Datenbank "Web-Navigator" und erstellt ein NotesDatabase-Objekt dafür.
  intErgebnis = NotesDatabase.OpenURLDb

- **OpenWithFailOver**
  öffnet eine Datenbank auf dem Server und erstellt ein NotesDatabase-Objekt. Gehört der Server zu einem Cluster, sucht Domino nach einer Replik der Datenbank auf einem anderen Server, wenn die angegebene Datenbank nicht geöffnet werden kann.
  intErgebnis = NotesDatabase.OpenWithFailOver(_
  stringServer, stringDateiname)

- **QueryAccess**
  gibt die Zugriffsberechtigung des aktuellen Benutzers auf die Datenbank zurück.
  intErgebnis = NotesDatabase.QueryAccess(stringBenutzer)

- **Remove**
  löscht die Datenbank von der Festplatte.
  Call NotesDatabase.Remove

- **Replicate**
  löst eine Replikation der Datenbanken mit ihren Repliken auf einem angegebenen Server aus.
  intErgebnis = NotesDatabase.Replicate(stringServer)

- **RevokeAccess**
  entfernt eine Person, Gruppe oder einen Server aus der Zugriffskontrollliste einer Datenbank. Damit wird die Zugriffsberechtigung auf „Default" zurückgesetzt.
  Call NotesDatabase.RevokeAccess(stringName)

- **Search**
  erstellt eine Dokumentensammlung anhand einer Abfrage. Es wird das Datum angegeben, ab dem die Suche beginnen soll und welche Anzahl an Dokumenten maximal in der Sammlung enthalten sein soll.
  Set NotesDocumentCollection = _
  NotesDatabase.Search(stringAbfrage, NotesDataTime, _
  intMaxDokumente)

- **UnprocessedFTSearch**
  erstellt eine Sammlung von NotesDocument-Objekten, die für einen Agenten als unverarbeitet gelten.
  Set NotesDocumentCollection = NotesDatabase(stringAbfrage, intMaxDokumente[,
  intSortierung[, intOptionen]])
  Für die Sortierung können folgende Konstante verwendet werden:

FT_SCORES	nach Vorkommen
FT_DATE_DES	nach Datum absteigend
FT_DATE_ASC	nach Datum aufsteigend

  Die Optionen können folgende Werte annehmen:

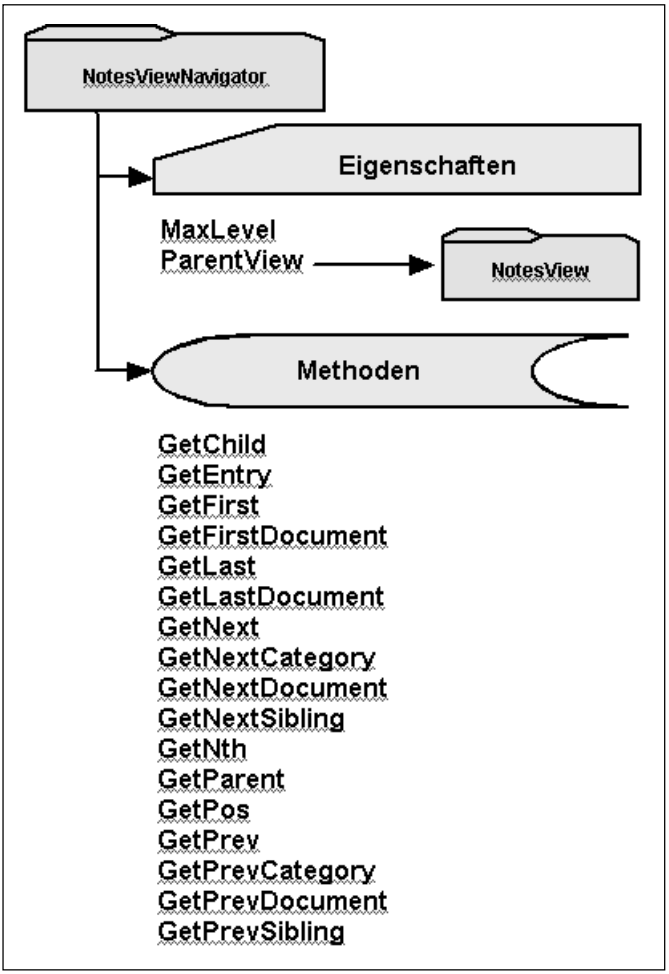
FT_STEMS	verwendet Worte als Basis für die Suche
FT_DATABASE	durchsucht Domino-Datenbanken
FT_FILESYSTEM	durchsucht auch Dateien, die keine Domino-Datenbanken sind

- **UnprocessedSearch**
  erstellt eine Sammlung von NotesDocument-Objekten, die nach einem bestimmten Datum geändert wurden und die einer Abfrage entsprechen.
  Set NotesDocumentCollection = _
  NotesDatabase.UnprocessedSearch(stringAbfrage, _
  NotesDateTime, intMaxDokumente)

- **UpdateFTIndex**
  bringt den Volltext-Index einer Datenbank auf den neuesten Stand.
  Call NotesDatabase.UpdateFTIndex(intAnlegen)

6.3.6.2.3 Die Klasse NotesViewNavigator

Die Klasse NotesViewNavigator hat große Ähnlichkeit mit der Klasse NotesViewEntryCollection. Der Unterschied besteht darin, dass in der Klasse NotesViewNavigator auch Kategorie- und Summenzeilen als Einträge enthalten sind.



Die Klasse NotesViewNavigator

Diese Eigenschaft der Klasse NotesViewNavigator lässt sich dafür einsetzen, einen Promptdialog nur mit Kategorie oder Summenzeilen zu füllen, ohne dafür eine eigene Ansicht erstellen zu müssen.

Die Klasseneigenschaften sind:

- **Maxlevel**
  Maximale Anzahl an Hierarchiestufen, aus denen Einträge in den NotesViewNavigator aufgenommen werden sollen.
  longMaxEbenen = _
  NotesViewNavigator.MaxLevel NotesViewNavigator.MaxLevel = _
  longMaxEbenen

- **ParentView**
  Die Eigenschaft „ParentView" enthält ein Objekt der Klasse NotesView, das der zugrundeliegenden Ansicht entspricht.
  NotesView = NotesViewNavigator.ParentView

Die Methoden der Klasse beziehen sich auf Einträge in der zugrundliegenden Ansicht. Die Methoden, die das Wort „Document" im Namen tragen, beziehen sich ebenfalls auf Ansichtseinträge und nicht auf Dokumente, dabei werden allerdings die Einträge für Kategorien und Summen nicht einbezogen. Folgende Methoden sind verfügbar.

- **GetChild**
  liefert einen Ansichtseintrag zurück, der eine Ebene darunter liegt. Gibt es mehrere Dokumente in dieser Ebene, wird der erste Eintrag verwendet. Gibt es keine Einträge in dieser Ebene, wird NULL zurückgegeben.
  NotesViewEntry = NotesViewNavigator.GetChild

- **GetEntry**
  erstellt ein neues NotesViewEntry-Objekt aus dem angegebenen NotesViewEntry-Objekt.
  NotesViewEntry = NotesViewNavigator.GetEntry(_
  NotesViewEntry)
  NotesViewNavigator.MaxLevel = longMaxEbenen

- **GetFirst, GetFirstDocument**
  Beide Methoden erzeugen ein NotesViewEntry-Objekt aus dem ersten Eintrag des NotesViewNavigators.
  NotesViewEntry = NotesViewNavigator.GetFirst

- **GetLast, GetLastDocument**
  erzeugen ein NotesViewEntry-Objekt aus dem letzten Eintrag des NotesViewNavigators.
  NotesViewEntry = NotesViewNavigator.GetLast

- **GetNext, GetNextDocument**
  liefern ein NotesViewEntry-Objekt aus dem Eintrag an der nächsten Position in der Ansicht.
  NotesViewEntry = _NotesViewNavigator.GetNext(NotesViewEntry)

- **GetNextCategory**
  GetNextCategory erstellt ein Objekt der Klasse „NotesViewEntry" aus der nächsten Kategoriezeile der Ansicht.
  NotesViewEntry = NotesViewNavigator.GetNextCategory

- **GetNextSibling**
  liefert den nächsten Eintrag eines abhängigen Dokumentes vom Dokument des angegebenen Eintrags.
  NotesViewEntry = NotesViewNavigator.GetNextSibling(_
  NotesViewEntry)

- **GetNth**
  erstellt ein NotesViewEntry-Objekt aus dem Eintrag mit der angegebenen Positionsnummer in der zugrundeliegenden Ansicht.
  NotesViewEntry = NotesViewNavigator.GetNth(longPosition)

- **GetParent**
  erzeugt ein NotesView-Objekt für die dem NotesViewNavigator zugrundeliegende Ansicht.
  NotesView = NotesViewNavigator.GetParent

- **GetPos**
  Die Methode „GetPos" verwendet eine Zeichenkette zur Definition der Position eines Eintrages in der zugrundeliegenden Ansicht. Die Ebenen werden mitnummeriert und die einzelnen Ebenenzahlen durch das angegebene Zeichen getrennt. Die Positionskennzahl kann z.B. folgendes Aussehen haben: 2.1.4. Es wird dann der 4. Eintrag in der ersten Unterebene der 2. Ebene ausgewählt und als NotesViewEntry-Objekt erstellt.
  NotesViewEntry = _
  NotesViewNavigator.GetPos(stringPosition, _
  stringTrennzeichen)

- **GetPrev, GetPrevDocument**
  liefern ein NotesViewEntry-Objekt aus dem Eintrag an der vorherigen Position in der Ansicht.
  NotesViewEntry = NotesViewNavigator.GetPrev

- **GetPrevCategory**
  GetPrevCategory erstellt ein Objekt der Klasse „NotesViewEntry" aus der vorherigen Kategoriezeile der Ansicht.
  NotesViewEntry = NotesViewNavigator.GetPrevCategory

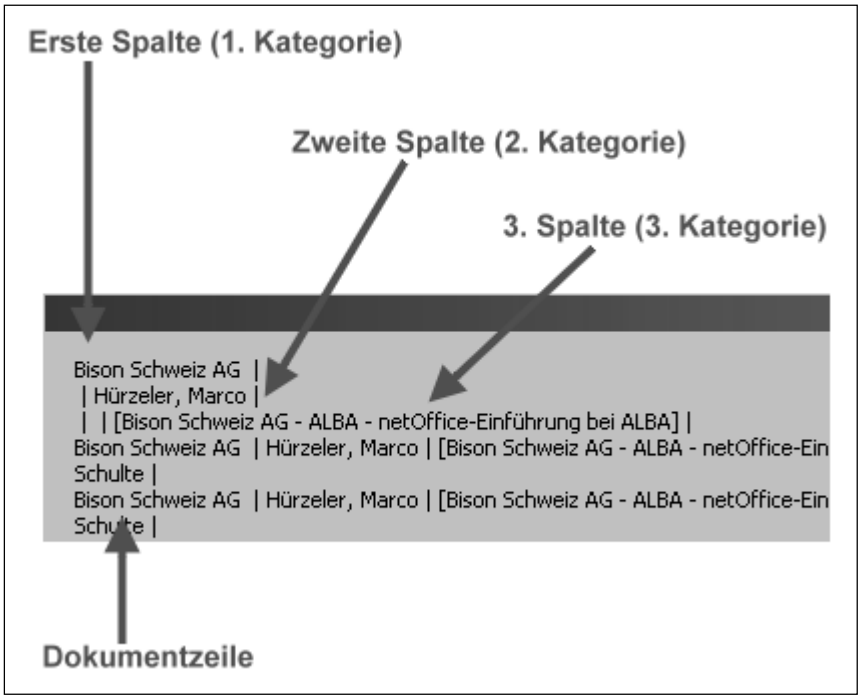
- **GetPrevSibling**
  liefert den vorherigen Eintrag eines abhängigen Dokumentes vom Dokument des angegebenen Eintrags.
  NotesViewEntry = NotesViewNavigator.GetPrevSibling(_ NotesViewEntry)

Programmtext

```
,/// Diese Prozedur wird über eine Schaltfläche gestartet
,/// und zeigt eine Liste der Einträge in der Ansicht an.
,/// Zur Kennzeichnung der einzelnen Spalten wurde jeweils ein
,/// „|"-Zeichen eingefügt.

Sub Click(Source As Button)
,///
    Dim ses As New NotesSession
    Dim ws As New NotesUiWorkspace
    Dim vw As NotesView
    Dim vn As NotesViewNavigator
    Dim ve As NotesViewEntry
    Dim doc As NotesDocument
    Dim uidoc As NotesUiDocument
    Dim strMsg As String

    ,/// NotesView-Objekt erstellen
    Set vw = ses.CurrentDatabase.getView("vwKorrAlpOff")

    ,/// NotesViewNavigator-Objekt aus allen Einträgen der
    ,/// Ansicht erzeugen
    Set vn = vw.createViewNav
```

```
    ,/// Ersten Eintrag des Navigators auswählen
    Set ve = vn.getFirst

    ,/// Alle Einträge verarbeiten
    While Not ve Is Nothing

      ,/// Alle Vorhandenen Spalten eines Eintrages
      ,/// verarbeiten.
      ,/// Achtung: Die Anzahl der Spalten kann
      ,/// unterschiedlich sein, weil die Kategorien
      ,/// nur einzelne Spalten betreffen.
      Forall c In ve.ColumnValues

          ,/// Nachrichtentext zusammensetzen
          strMsg = strMsg & c & " | "
      End Forall

      ,/// Nachricht um eine Zeilenschaltung ergänzen
      strMsg = strMsg & Chr$(10)

      ,/// Nächsten Eintrag auswählen
      Set ve = vn.getNext(ve)
    Wend

    ,/// Nachricht am Bildschirm anzeigen
    Msgbox strMsg

,///
End Sub
```

Der folgende Ausschnitt aus der Nachricht zeigt die Kategoriezeilen mit den darin enthaltenen Spalten. Darunter befinden sich dann die Einträge für die Dokumente.



Anzeige eines NotesViewNavigators

Eine besonders interessante Möglichkeit steckt in den „Create"-Methoden des NotesView-Objektes: Diese Methoden erlauben es, NotesViewNavigator-Objekte aus einer Teilmenge der Einträge der Ansicht zu erstellen. Dabei sind folgende Methoden verfügbar:

- **CreateViewNavFrom**
  Bei dieser Methode werden die Einträge einer Ansicht ausgewählt, beginnend mit dem angegebenen Eintrag bis zum letzten Eintrag der Ansicht.
  NotesViewNavigator = _
  NotesView.CreateViewNavFrom(NotesViewEntry oder NotesDocument)

- **CreateViewNavFromCategory**
  erzeugt ein NotesViewNavigator-Objekt aus den Einträgen der Ansicht zu einer bestimmten Kategorie.
  NotesViewNavigator = _
  NotesView.CreateViewNavFromCategory(_
  stringKategorie)

- **CreateViewNavFromChildren**
  wählt die unmittelbar abhängigen Einträge von dem angegebenen Eintrag oder Dokument aus und erstellt daraus ein NotesViewNavigator-Objekt.
  NotesViewNavigator = _
  NotesView.CreateViewNavFromChildren(_
  NotesViewEntry oder NotesDocument)

- **CreateViewNavFromDescendants**
  verwendet alle abhängigen Einträge vom angegebenen Eintrag in der Ansicht zur Erstellung des NotesViewNavigator-Objektes.
  NotesViewNavigator = _
  NotesView.CreateViewNavFromDescendants(_
  NotesViewEntry oder NotesDocument)

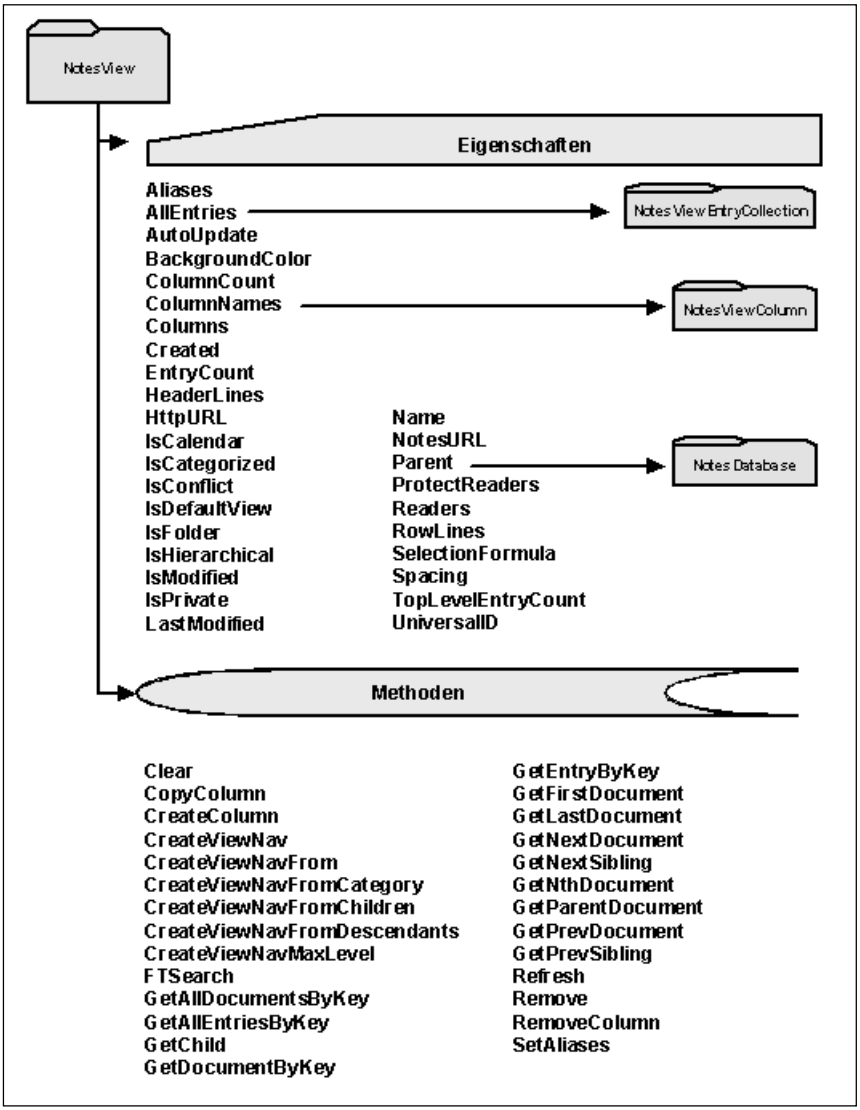
Das auf diese Weise entstandene NotesViewNavigator-Objekt kann jetzt dazu verwendet werden, nur die in ihm aufgeführten Dokumente zu verarbeiten. Der Vorteil dieser Vorgehensweise besteht darin, eine genau definierte Anzahl von Dokumenten zu erhalten, ohne ständig eine entsprechende Ansicht in der Datenbank mitzuführen.

6.3.6.2.4 Die Klasse NotesView

Zur Verwaltung von Datensätzen durch Auswahl und Sortierfolge verwendet Lotus Domino Ansichten und Ordner. Aus der Sicht von LotusScript sind Ordner lediglich eine spezielle Form von Ansichten (Ansichten ohne SELECT-Formel) und werden einfach als Ansichten (NotesView) definiert. Über die Ansichten werden zwei Funktionen realisiert:

- die Darstellung von Daten am Bildschirm
- der Zugriff auf Dokumente

Für die Verwendung von Ansichten als Benutzeroberfläche wurde in LotusScript die Klasse „NotesUiView" bei den Frontend-Klassen definiert. Der Zugriff auf Dokumente ist die wichtigste Aufgabe der Klasse „NotesView".



Die Klasse NotesView

Neben den Methoden werden in den Eigenschaften immer häufiger Möglichkeiten geschaffen, durch Schreibzugriffe das Aussehen und den Inhalt der Ansichten per Programm zu verändern. Es ergeben sich durch das dynamische Ändern von Ansichten viele Möglichkeiten, unterschiedliche Dokument-Auswahlen zu verwenden, ohne für jede Variante eine neue Ansicht zu definieren, die den Umfang einer Datenbank aufbläht.

Folgende Eigenschaften sind für die Klasse NotesView verfügbar:

- **Aliases**
  gibt die Alias-Namen einer Ansicht zurück. Mit Hilfe der Methode "SetAliases" kann die Aliases-Eigenschaft überschrieben werden.
  stringAlias = NotesView.Aliases

- **AllEntries**
  Die Einträge einer Ansicht enthalten alle Spaltenwerte, die in der Ansicht definiert sind. Die Werte sind verfügbar, ohne dass das dazugehörige Dokument noch einmal geöffnet werden muss. Der Zugriff auf die Ansichtseinträge erfolgt über die Klasse NotesViewEntryCollection.
  NotesViewEntryCollection = NotesView.AllEntries

Programmtext

```
Function wPrompt as Variant
'///
'/// Die Funktion ermittelt aus einer Ansicht
'/// ein Array mit den Werten einer Spalte
'/// und liefert sie als Variantvariable an das
'/// aufrufende Programm zurück.
'///
    Dim ses As New NotesSession
    Dim vw As NotesView
    Dim vc As NotesViewEntryCollection
    Dim ve As NotesViewEntry
    Dim arListe() As String
    Dim varListe As Variant
    Dim i As Integer
    Dim varZeile As Variant

    '/// Ansichtsobjekt erstellen
    Set vw = ses.CurrentDatabase.GetView("vwGruppe")

    '/// NotesViewEntryCollection-Objekt mit allen
    '/// Zeilen der Ansicht erstellen
    Set vc = vw.AllEntries

    '/// NotesViewEntry-Objekt mit der ersten
    '/// Zeile erstellen
    Set ve = vc.GetFirstEntry
    i=0
```

```
    Do While Not ve Is Nothing

      '/// Wenn in der ersten Spalte der Ansicht
      '/// der Name des aktuellem Benutzers steht,
      '/// wird der Wert der 4. Spalte in ein
      '/// Element eines Arrays übernommen

      If ve.ColumnValues(0) = ses.UserName Then
            '/// Neaues Array-Element anlegen
            Redim Preserve arliste(i) As String
            arListe(i) = ve.ColumnValues(3)
            i = i + 1
      End If

      '/// Objekt für die nächste Zeile erstellen
      Set ve = vc.GetNextEntry(ve)
    Loop

    If i = 0 Then
      '/// Wenn kein Wert in die Liste übernommen
      '/// wurde: Funktion beenden.
      wPrompt = NULL
    Else
      '/// Liste zurückgeben
      wPrompt = arListe
    End If

'///
End Function
```

- **AutoUpdate**
  Diese Eigenschaft zeigt an, ob die Ansicht automatisch aktualisiert wird, wenn ein Dokument erstellt, geändert oder gelöscht wird, dessen Feldwerte der Ansichtsauswahl entsprechen. Die Eigenschaft kann gelesen und geändert werden.
  boleanVar = NotesView.AutoUpdate
  NotesView.AutoUpdate = True

- **BackgroundColor**
  Der Hintergrund einer Ansicht kann eine Farbe annehmen. Diese Farbe kann ermittelt und verändert werden. Die Farbe wird durch einen Zahlenwert von 0 bis 240 dargestellt.
  integerVar = NotesView.BackgroundColor
  NotesView.BackgroundColor = integerVar

- **ColumnCount**
  gibt die Anzahl der Spalten einer Ansicht aus.
  integerVar = NotesView.ColumnCount

- **Columns**
  Die Eigenschaft „Columns" enthält Objekte der Klasse „NotesViewColumn" in Form eines Arrays. Der Zugriff erfolgt über die einzelnen Elemente der Eigenschaft oder über eine Variantvariable.
  NotesViewColumn = NotesView.Columns(integerElement)

- **Created**
  enthält das Datum, an dem die Ansicht erstellt wurde.
  variantDatum = NotesView.Created

- **HeaderLines**
  Anzahl der Überschriftzeilen für eine Ansicht.
  integerZeilen = NotesView.HeaderLines

- **IsCalendar**
  Notes-Ansichten kennen zwei Darstellungsformen: die Standardansicht und die Kalenderansicht. Die Eigenschaft „IsCalendar" besagt, ob die Ansicht in Form eines Kalenders angezeigt wird.
  booleanVar = NotesView.IsCalendar

- **IsCategorized**
  Zeigt an, ob die Ansicht eine kategorisierte Spalte enthält.
  booleanVar = NotesView.IsCategorized

- **IsConflict**
  In Kalenderansichten kann geprüft werden, ob ein Terminkonflikt vorliegt. Die Eigenschaft „IsConflict" enthält die Aussage darüber, ob auf Konflikte geprüft wird.
  booleanVar = NotesView.IsConflict

- **IsDefaultView**
  Jede Datenbank benötigt wenigstens eine Ansicht. Das liegt daran, dass Notes beim Öffnen der Datenbank einen Bezug zwischen den Daten und deren Verarbeitung schaffen muss. Wird keine Angabe dazu gemacht, welche Ansicht in der Datenbank verwendet werden soll, greift Notes auf die „Vorgabeansicht" zurück. Diese Ansicht kann in den Ansichtseigenschaften festgelegt werden. Ist keine Ansicht als Standardansicht gekennzeichnet, wird die erste Ansicht in der Liste der vorhandenen Ansichten verwendet. Die Eigenschaft „IsDefaultView" kann gelesen und geändert werden.
  booleanVar = NotesView.IsDefaultView
  NotesView.IsDefaultView = booleanVar

- **IsFolder**
  Handelt es sich bei einer Ansicht um einen Ordner, so ist die Eigenschaft „IsFolder" auf „True" gesetzt.
  booleanVar = NotesView.IsFolder

- **IsHierarchical**
  gibt zurück, ob eine Ansicht die Antwortdokumente und Antworten auf Antwortdokumente mit dem dazugehörigen Hauptdokument zusammen anzeigt.
  booleanVar = NotesView.IsHierarchical

- **IsModified**
  Da in einer Client-/Server-Architektur ein zeitlicher Unterschied zwischen der Anzeige einer Ansicht und dem echten Inhalt einer Ansicht auf dem Server bestehen kann, zeigt die Eigenschaft „IsModified" an, ob ein Unterschied zwischen den Ansichten auf Client und Server besteht. Ist die Eigenschaft „True", sollte die Anzeige der Ansicht aktualisiert werden.
  booleanVar = NotesView.IsModified

- **IsPrivate**
  gibt an, ob es sich um eine Ansicht des aktuellen Benutzers handelt, die nicht für andere Benutzer zur Verfügung steht.
  booleanVar = NotesView.IsPrivate

- **LastModified**
  enthält das Datum der letzten Änderung der Ansicht in Form einer Variantvariablen.
  variantVar = NotesView.LastModified

- **Name**
  Der Name einer Ansicht kann über diese Eigenschaft ermittelt und verändert werden.
  stringVar = NotesView.Name
  NotesView.Name = stringVar

- **Parent**
  enthält ein NotesDatabase-Objekt der Datenbank, in der die Ansicht gespeichert ist.
  NotesDatabase = NotesView.Parent

Programmtext

```
'/// Beim Anklicken einer Schaltfläche werden alle Eigenschaften
'/// einer Ansicht angezeigt, die über "Is..." abgefragt werden
'/// können.

Sub Click
'///
    Dim ses as New NotesSession
    Dim vw as NotesView
    Dim strMsg as String

    '/// NotesView-Objekt erstellen
    Set vw = ses.CurrentDatabase.GetView("vwAdrName")

    '/// Eigenschaften der Ansicht zusammenstellen
    strMsg = "Die Ansicht " & vw.Name & _
      " hat folgende Eigenschaften: " & _
      chr$(10) & chr$(10)
```

```
If vw.IsCalendar then
  strMsg = strMsg & _
       "Es handelt sich um eine Kalenderansicht" _
       & chr$(10)
Else
  strMsg = strMsg & _
       "Es handelt sich um eine Standardansicht" _
       & chr$(10)
End If

If vw.IsCategorized then
  strMsg = strMsg & _
       "Die Ansicht verfügt über Kategorien" & _
       chr$(10)
Else
  strMsg = strMsg & _
       "Die Ansicht verfügt nicht " & _
       "über Kategorien" & chr$(10)
End If

If vw.IsConflict then
  strMsg = strMsg & _
       "Die Terminkonfliktprüfung ist aktiviert" & _
       chr$(10)
Else
  strMsg = strMsg & _
       "Die Terminkonfliktprüfung ist nicht " _
       & " aktiviert" & chr$(10)
End If
```

```
If vw.IsDefaultView then
  strMsg = strMsg & _
       "Es handelt sich um die Vorgabeansicht" _
       & chr$(10)
Else
  strMsg = strMsg & _
       "Es handelt sich nicht um die " _
       & "Vorgabeansicht" & chr$(10)
End If

If vw.IsFolder then
  strMsg = strMsg & _
       "Es handelt sich um einen Ordner" & _
       chr$(10)
Else
  strMsg = strMsg & _
       "Es handelt sich um eine Ansicht" & _
       chr$(10)
End If

If vw.IsHierarchical then
  strMsg = strMsg & _
       "Die Ansicht zeigt Antwortdokumente an" _
       & chr$(10)
Else
  strMsg = strMsg & _
       "Die Ansicht zeigt Antwortdokumente " _
       & "nicht an" & chr$(10)
End If
```

```
    If vw.IsModified then
      strMsg = strMsg & _
           "Die Ansicht wurde geändert" & _
           chr$(10)
    Else
      strMsg = strMsg & _
           "Die Ansicht wurde nicht geändert" & _
           chr$(10)
    End If

    If vw.IsPrivate then
      strMsg = strMsg & _
           "Die Ansicht ist privat" & _
           chr$(10)
    Else
      strMsg = strMsg & _
           "Die Ansicht ist öffentlich" & _
           chr$(10)
    End If

    '/// Eigenschaften in einem Nachrichtenfeld anzeigen
    MsgBox strMsg, 64, "Ansicht " & vw.Name

'///
End Sub
```

Die Nachricht hat dann folgendes Aussehen:

Ansichtseigenschaften

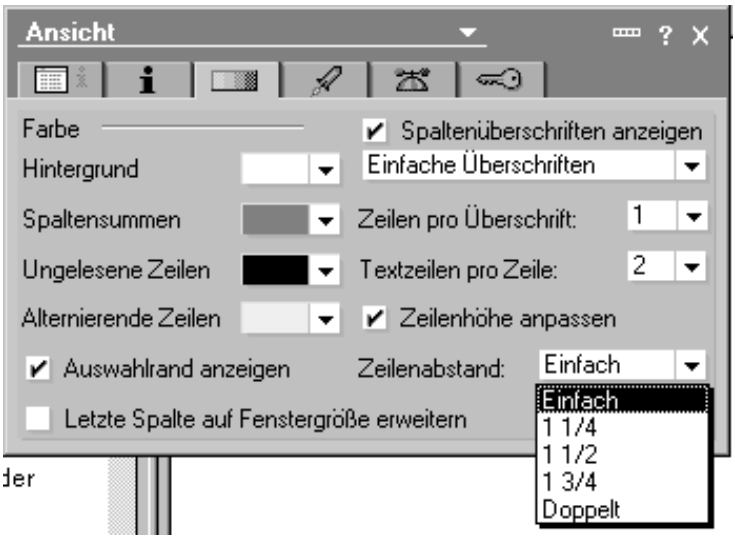
- **ProtectReaders**
  In den Leser-Eigenschaften der Ansicht ist hinterlegt, wer sie anzeigen darf. Über die Eigenschaft „ProtectReaders" wird gesteuert, ob die Leser-Eigenschaften bei einer Replikation verändert werden dürfen oder nicht. „ProtectReaders" kann gelesen und geändert werden.
  booleanVar = NotesView.ProtectReaders
  Notes.ProtectReaders = booleanVar

- **Readers**
  gibt die Namen der Personen wieder, die eine Ansicht anzeigen dürfen. Die Eigenschaft kann gelesen und geändert werden.
  stringArray = NotesView.Readers
  NotesView.Readers = StringArray

- **RowLines**
  Anzahl der Zeilen die für die Anzeige eines Dokumentes in der Ansicht verwendet werden können.
  integerVar = NotesView.RowLines

- **Spacing**
  gibt den Abstand zwischen zwei Zeilen einer Ansicht an. Die Eigenschaft kann gelesen und geändert werden. Anstelle der Zahlen können folgende Konstanten eingesetzt werden:
  - VW_SPACING_SINGLE – Einfach
  - VW_SPACING_ONE_POINT_25 – 1 1/4-zeilig
  - VW_SPACING_ONE_POINT_50 – 1 1/2-zeilig
  - VW_SPACING_ONE_POINT_75 – 1 3/4-zeilig
  - VW_SPACING_DOUBLE – Doppelt

Das entspricht den Werten, die in dem Eigenschaftsfenster der Ansicht definiert werden können.



Zeilenabstand in Ansichten

integerAbstand = NotesView.Spacing
NotesView.Spacing = integerAbstand

- **TopLevelEntryCount**
  Anzahl der Einträge für die erste Kategoriespalte der Ansicht.
  integerAnzahl = NotesView.TopLevelEntryCount

- **UniversalID**
  Die UniversalID dient als eindeutiger Schlüssel für die Identifizierung der Ansicht.
  Es handelt sich dabei um eine Zeichenkette mit Buchstaben und Zahlen mit 32 Zeichen.
  stringVar = NotesView.UniversalID

Die Methoden der NotesView-Klasse werden für die Arbeit mit Dokumenten, Spalten und Navigatoren verwendet.

- **Clear**
  entfernt den Volltext-Index aus der Ansicht.
  Call NoteView.Clear

- **CreateViewNav, CreateViewNavFrom, CreateViewNavFromCategory, CreateViewNavFromChildren, CreateViewNavFromDescendants**
  Diese Methoden dienen der Erzeugung von Navigator-Objekten aus einer bestehenden Ansicht.

- **FTSearch**
  filtert eine Ansicht in der Form, dass sie nur noch die Dokumente anzeigt, die der Abfrage entsprechen, die angegeben wird. Zusätzlich kann eine Obergrenze der ausgewählten Dokumente angegeben werden. Als Ergebnis erhält man die Anzahl der ausgewählten Dokumente.
  longAnzahl = NotesView.FTSearch(stringAbfrage, _ integerMaxDokumente)

- **GetAllDocumentsByKey**
  Mit Hilfe eines Arrays mit Schlüsselbegriffen wird eine Anzahl von Dokumenten aus der Ansicht ausgewählt und einem Objekt der Klasse „NotesDocumentCollection" übergeben.
  Set NotesDocumentCollection =
  NotesView.GetAllDocumentsByKey(Array _
  [, booleanExakt])

- **GetAllEntriesByKey**
  Mit Hilfe eines Arrays mit Schlüsselbegriffen wird eine Anzahl von Einträgen aus der Ansicht ausgewählt und einem Objekt der Klasse „NotesViewEntryCollection" übergeben.
  Set NotesViewEntryCollection = _
  NotesView.GetAllEntriesByKey(Array _
  [, booleanExakt])

- **GetChild**
  ermittelt das erste Antwortdokument zu dem angegebenen Dokument aus einer Ansicht. Als Parameter wird der Methode das Hauptdokument als Objekt übergeben.
  Set NotesDocument = NotesView.GetChild(_
  NotesDocument)

- **GetDocumentByKey**
  Auswahl eines Dokumentes als Objekt der Klasse „NotesDocument" über ein Array mit Schlüsselbegriffen.
  Set NotesDocument = NotesView.GetDocumentByKey(_
  Array[, booleanExakt])

Programmtext

```
Sub getDokAdresse(strDocID As String)
‘///
   Dim ses As New NotesSession
   Dim dbAdr As NotesDatabase
   Dim vwAdr As NotesView
   Dim docAdr As NotesDocument

   ‘/// NotesDatabase-Objekt für die Adressdatenbank
   ‘/// erzeugen
   Set dbAdr = New _
     NotesDatabase(ses.CurrentDatabase.Server, _
     “Adressen.nsf”)

   ‘/// NotesView-Objekt erzeugen
   Set vwAdr = dbAdr.GetView(“vwKundenDocID”)

   ‘/// Mit Hilfe der Methode “GetDocumentByKey”
   ‘/// das passende NotesDocument-Objekt erstellen
   Set docAdr = vwAdr.GetDocumentByKey(strDocID)

   If docAdr Is Nothing Then
     Print “Person nicht gefunden!”
   Else
     Print docAdr.FirstName(0) & “ “ & _
          docAdr.LastName(0) & “ gefunden!”
   End If

‘///
End Sub
```

- **GetEntryByKey**
  Auswahl eines Eintrages aus einer Ansicht als Objekt der Klasse „NotesViewEntry" über ein Array mit Schlüsselbegriffen.
  Set NotesViewEntry = NotesView.GetEntryByKey(_
  Array[, booleanExakt])

Programmtext

```
'/// Die Prozedur "getDokAdresse" sucht in einer Ansicht den
'/// Eintrag, der zu dem angegebenen Schlüssel passt. Ohne das
'/// dazugehörige Dokument zu öffnen, werden Vor- und Nachname
'/// in der Statuszeile ausgegeben.

Sub getDokAdresse(strDocID As String)
'///
    Dim ses As New NotesSession
    Dim dbAdr As NotesDatabase
    Dim vwAdr As NotesView
    Dim veAdr As NotesViewEntry

    '/// NotesDatabase-Objekt für die Adressdatenbank
    '/// erzeugen
    Set dbAdr = New _
      NotesDatabase(ses.CurrentDatabase.Server, _
      "Adressen.nsf")

    '/// NotesView-Objekt erzeugen
    Set vwAdr = dbAdr.GetView("vwKundenDocID")
```

```
    '/// Mit Hilfe der Methode "GetEntryByKey"
    '/// das passende NotesViewEntry-Objekt erstellen
    Set veAdr = vwAdr.GetEntryByKey(strDocID)

    If veAdr Is Nothing Then
      Print "Person nicht gefunden!"
    Else
      '/// In der zweiten Spalte der Ansicht ist der
      '/// Nachname und in der dritten Spalte der
      '/// Vorname enthalten.
      Print veAdr.ColumnValues(2) & " " & _
            veAdr.ColumnValues(1) & " gefunden!"
    End If

'///
End Sub
```

- **GetFirstDocument**
  Diese Methode ermittelt das erste Dokument einer Ansicht als Objekt der Klasse „NotesDocument".
  Set NotesDocument = NotesView.GetFirstDocument

Programmtext

```
'/// Die Prozedur "PrintAdress" zeigt die Namen aller Personen
'/// deren Adressdokument in der Ansicht "vwLastName"
'/// enthalten ist.

  Sub PrintAdress
  '///
    Dim ses As New NotesSession
  Dim dbAdr As NotesDatabase
  Dim vwAdr As NotesView
  Dim veAdr As NotesViewEntry

  '/// NotesDatabase-Objekt für die Adressdatenbank
  '/// erzeugen
  Set dbAdr = New _
    NotesDatabase(ses.CurrentDatabase.Server, _
    "Adressen.nsf")

  '/// NotesView-Objekt erzeugen
  Set vwAdr = dbAdr.GetView("vwLastName")

  '/// NotesDocument-Objekt aus dem ersten Eintrag der
  '/// Ansicht erstellen
  Set docAdr = vwAdr.GetFirstDocument
```

```
    '/// Alle Namen in der Statuszeile anzeigen. Die Schleife wird
    '/// solange ausgeführt, bis kein Eintrag mehr gefunden wird.
    '/// docAdr enthält dann einen NULL-Wert, der mit
    '/// "is Nothing" abgefragt werden kann.
    While Not docAdr is Nothing
      Print docAdr.LastName(0) & ", " & _
            docAdr.FirstName(0)

      '/// Objekt für nächsten Eintrag erzeugen
      Set docAdr = vwAdr.GetNextDocument(docAdr)
    Wend

'///
End Sub
```

- **GetLastDocument**
  Diese Methode ermittelt das letzte Dokument einer Ansicht als Objekt der Klasse „NotesDocument".
  Set NotesDocument = NotesView.GetFirstDocument

- **GetNextDocument**
  Ist das erste Dokument einer Ansicht ermittelt, so können die weiteren Dokumente im Rahmen einer Schleife über die Methode „GetNextDocument" gefunden werden. Dabei wird jeweils das bekannte Dokument als Parameter übergeben.
  Set NotesDocument = NotesView.GetNextDocument(_
  NotesDocument)

- **GetNextSibling**
  Die Methode „GetNextSibling" unterscheidet zwischen Haupt- und Antwortdokumenten. Sie liefert das nächste Element der gleichen Ebene (also ausgehend von einem Hauptdokument das nächste Hauptdokument und ausgehend von einem Antwortdokument das nächste Antwortdokument).
  Set NotesDocument = NotesView.GetNextSibling(_ NotesDocument)

- **GetNthDocument**
  Diese Methode ermittelt das n-te Dokument einer Ansicht als Objekt der Klasse „NotesDocument". Damit ist es möglich, über eine For-Next-Schleife auf die Dokumente einer Ansicht zuzugreifen.
  Set NotesDocument = NotesView.GetNthDocument(_ longElement).

Programmtext

```
'/// Die Prozedur "ShowPersons" übernimmt alle Dokumente der
'/// Adressdatenbank in eine NotesDocumentCollection und zeigt
'/// die Namen aller Adressen an, bei denen die angegebene
'/// Funktion zutrifft.

Sub ShowPersons(strFunktion as String)
'///
    Dim ses As New NotesSession
    Dim dbAdr As NotesDatabase
    Dim colAdr As NotesDocumentCollection
    Dim docAdr as NotesDocument
    Dim i as Integer
```

```
    '/// NotesDatabase-Objekt für die Adressdatenbank
    '/// erzeugen
    Set dbAdr = New _
      NotesDatabase(ses.CurrentDatabase.Server, _
      "Adressen.nsf")

    '/// NotesDocumentCollection-Objekt erstellen
    Set colAdr = dbAdr.AllDocuments

    For i = 1 to colAdr.count
      '/// NotesDocument-Objekt erzeugen
      Set docAdr = colAdr.GetNthDocument(i)

      '/// Namen anzeigen
      If docAdr.Funktion(0) = strFunktion then
            Print docAdr.LastName(0) & ", " & _
                  docAdr.FirstName(0)
      End If
    Next i
'///
End Sub
```

- **GetParentDocument**
  liefert zu einem Antwortdokument einer Ansicht das dazugehörige Hauptdokument.
  Set NotesDocument = NotesView.GetParentDocument(_ NotesDocument)

- **GetPrevDocument**
  ermittelt das vorherige Dokument zum aktuellen Dokument als Objekt.
  Set NotesDocument = NotesView.GetPrevDocument(_ NotesDocument)

- **GetPrevSibling**
  Die Methode "GetPrevSibling" unterscheidet zwischen Haupt- und Antwortdokumenten. Sie liefert das davor liegende Element der gleichen Ebene (also ausgehend von einem Hauptdokument das nächste Hauptdokument und ausgehend von einem Antwortdokument das nächste Antwortdokument).
  Set NotesDocument = NotesView.GetPrevSibling(_ NotesDocument)

- **Refresh**
  Wird eine Ansicht eingelesen, so bleibt sie am Bildschirm unverändert, auch wenn sich die Dokumente auf der Festplatte verändern. Um die Ansicht auf den aktuellen Stand zu bringen, verwendet man die Methode "Refresh".
  Call NotesView.Refresh

Programmtext

```
‘/// Die Prozedur “getMemos” übernimmt alle Dokumente aus
‘/// dem angegebenen Ordner in die aktuelle
‘/// Korrespondenzdatenbank und fügt zusätzlich Felder ein, die
‘/// in der Korrespondenzdatenbank erforderlich sind.

Sub getMemos(strFolder As String)
‘///
    Dim ses As New NotesSession
    Dim ws As New NotesUiWorkSpace
    Dim dbMail As NotesDatabase
    Dim vw As NotesView
    Dim vwMail As NotesView
    Dim docMail As NotesDocument
    Dim docOffice As NotesDocument
    Dim item As NotesItem
    Dim rtitem As NotesRichTextItem
    Dim intCount As Integer
    Dim strMailFile As String
    Dim strMsg As String

    ‘/// Name der Mail-Datenbank aus der Datei “Notes.ini”
    ‘/// ermitteln
    strMailFile = Cstr(ses.GetEnvironmentString(“MailFile”, True))

    ‘/// NotesDatabase-Objekt für die Maildatenbank erzeugen
    Set dbMail = New _
      NotesDatabase(ses.CurrentDatabase.Server, _
      strMailFile)
```

```
‘/// NotesView-Objekt für den angegebenen Ordner erstellen
Set vwMail = dbMail.GetView(strFolder)

‘/// Prozedur verlassen, wenn kein Ordner mit dem
‘/// angegebenen gefunden wurde
If vwMail Is Nothing Then
  strMsg = “Ungültiger Ordnername!”  & Chr(10) & _
       Chr(10) & “Der Ordner mit dem Namen” & _
       Chr(10) & Chr(10) & _
       strFolder & Chr(10) & Chr(10) & _
       “fehlt in Ihrer Maildatenbank!”
  Msgbox strMsg, 16, “Fehler”
  Exit Sub
End If

intCount = 0
‘/// NotesDocument-Objekt für den ersten Eintrag in dem
‘/// Ordner erzeugen
Set docMail = vwMail.GetFirstDocument

While Not docMail Is Nothing
  ‘/// Neues NotesDocument-Objekt für das Dokument
  ‘/// erstellen, das in der aktuellen Datenbank erzeugt
  ‘/// werden soll.
  Set docOffice = _
       ses.CurrentDatabase.CreateDocument

  With docOffice
       ‘/// Maske zuweisen und Felder übernehmen
       .Form = “frmBrief”
       Set item = docMail.GetFirstItem(“Subject”)
```

```
If Not item Is Nothing Then
        item.CopyItemToDocument _
                docOffice, “Betreff”
        item.CopyItemToDocument _
                docOffice, “Subject”
End If

.FormularName = “Memo”
.strStatus = “Offen”
.DokArt = “E-Mail”
.Firmenzeile = “>>> Neues E-Mail <<<“
Set item = docMail.GetFirstItem(“Principal”)

If Not item Is Nothing Then
        item.CopyItemToDocument _
                docOffice, “From”
        strText = Lcase(item.Values(0))
        .Personenzeile = strText
        .MailAdress = strText
        .SendTo = strText
Else
        Set item = _
                docMail.GetFirstItem(“From”)
        item.CopyItemToDocument _
                docOffice, “NotAutor”
        item.CopyItemToDocument _
                docOffice, “From”
```

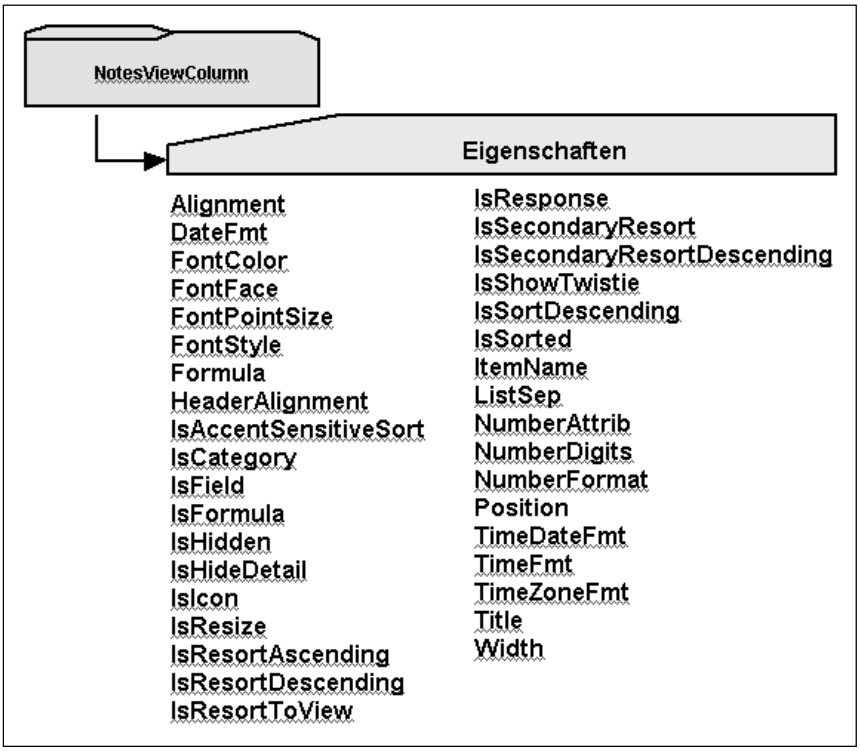
```
                strText = Lcase(item.Values(0))
                .Personenzeile = strText
                .EMailSB = strText
                .SendTo = strText
            End If

            .datWriteDay = docMail.PostedDate
            .Bearbeiter = ses.CommonUserName
            .Principal = ses.UserName
            .FolgeAktion = "Bearbeiten"
            Set rtitem = docMail.GetFirstItem("Body")

            If Not item Is Nothing Then
                rtitem.CopyItemToDocument _
                    docOffice, "Body"
            End If

                '/// Wenn alle Felder übernommen sind, das
            '/// neue NotesObjekt speichern
            .save True, False
        End With

            intCount = intCount + 1

            '/// NotesDocument-Objekt für den nächsten
        '/// Ordner-Eintrag erzeugen
        Set docMail = vwMail.GetNextDocument(docMail)

            '/// In der Statuszeile anzeigen, welches
        '/// Mail-Dokument gerade kopiert wird
        Print intCount; " E-Mails übernommen "
    Wend
```

```
        '/// In der aktuellen Datenbank wurden Dokumente
      '/// gespeichert, die in der am Bildschirm gezeigten Ansicht
      '/// nicht enthalten sind. Damit diese Ansicht wieder aktualisiert
      '/// wird, wird die Methode "ViewRefresh" ausgeführt.
      ws.ViewRefresh
  '///
  End Sub
```

- **Remove**
  Die Methode „Remove" löscht eine Ansicht komplett. Sie ist dann auf der Festplatte (in der Datenbank) nicht mehr verfügbar.
  Call NotesView.Remove

6.3.6.2.5 Die Klasse NotesViewColumn

Zusätzlich zur ganzen Ansicht lässt sich auch jede Spalte als einzelnes Objekt einsetzen. Es gibt zwar zur Zeit keine Methoden für diese Klasse, dafür aber sehr viele Eigenschaften. Leider können diese Eigenschaften nur gelesen und nicht verändert werden.



Die Klasse NotesViewColumn

Folgende Eigenschaften sind für die NotesViewColumn-Klasse verfügbar:

- **Alignment**
  Gibt die Textausrichtung der Spalte in Form einer Integer-Konstanten zurück.
  integerRichtung = NotesViewColumn.Alignment

  Gültige Werte sind:
  VC_ALIGN_LEFT – linksbündig – (0)
  VC_ALIGN_RIGHT – rechtsbündig – (1)
  VC_ALIGN_CENTER – zentriert – (2)

- **DateFmt**
  Diese Eigenschaft enthält das Datumsformat einer Spalte.
  integerFormat = NotesViewColumn.DateFmt

  Mögliche Werte sind:
  VC_FMT_YMD – Jahr, Monat und Tag – (0)
  VC_FMT_MD – Monat und Tag – (2)
  VC_FMT_YM – Jahr und Monat – (3)
  VC_FMT_Y4M – Jahr vierstellig und Monat – (6)

- **FontColor, FontFace, FontPointSize, FontStyle**
  sind die Eigenschaften des Textes einer Spalte. Die Farbe erhält man mit:
  integerFarbe = NotesViewColumn.FontColor

  Möglich sind die Integer-Konstanten:
  COLOR_BLACK – (0)
  COLOR_WHITE – (1)
  COLOR_RED – (2)
  COLOR_GREEN – (3)
  COLOR_BLUE – (4)

COLOR_MAGENTA – (5)
COLOR_YELLOW - (6)
COLOR_CYAN – (7)
COLOR_DARK_RED – (8)
COLOR_DARK_GREEN – (9)
COLOR_DARK_BLUE – (10)
COLOR_DARK_MAGENTA – (11)
COLOR_DARK_YELLOW – (12)
COLOR_DARK_CYAN – (13)
COLOR_GREY – (14)
COLOR_LIGHT_GRAY – (15)

Die Schriftart, wie z.B. „Arial", wird mit FontFace ermittelt:
stringSchrift = NotesViewColumn.FontFace

Die Schriftgröße in Punkt erhält man mit:
integerGroesse = NotesViewColumn.FontPointSize

Der Schriftstil, also z.B. „kursiv" enthält die „FontStyle"-Eigenschaft:
integerStil = NotesViewColumn.FontStyle

Die Integer-Konstanten für den Schriftstil sind:
VC_FONT_BOLD – fett – (1)
VC_FONT_ITALIC – kursiv – (2)
VC_FONT_UNDERLINE – unterstrichen – (4)
VC_FONT_STRIKEOUT – durchgestrichen – (8)

- **Formula**
  Da eine Spalte statt eines Feldinhaltes auch berechnete Werte enthalten kann, lässt sich in der Eigenschaft „Formula" die Berechnungsformel der Spalte ermitteln.
  stringFormel = NotesViewColumn.Formula

Bei einer Spalte ohne Formel (also mit einem Feld) enthält die Eigenschaft einen leeren String. Wird die Spalte über eine einfache Aktion berechnet, so enthält die „Formula"-Eigenschaft die Formel, die der einfachen Aktion entspricht. Wird z.B. die einfache Aktion „Anhänge" verwendet, so gibt die „Formula"-Eigenschaft den Wert „@Attachments" zurück.

- **HeaderAlignment**
  Ausrichtung der Spaltenüberschrift.
  integerRichtung = NotesViewColumn.HeaderAlignment

  Wie bei der Spalte sind folgende Werte möglich:
  VC_ALIGN_LEFT – linksbündig – (0)
  VC_ALIGN_RIGHT – rechtsbündig – (1)
  VC_ALIGN_CENTER – zentriert – (2)

- **IsAccentSensitiveSort, IsCaseSensitiveSort, IsResortAscending, IsResortDescending, IsResortToView, IsSecondarySort, IsSecondaryResortDescending, IsSortDescending, IsSorted**
  Diese Eigenschaften geben die verschiedenen Formen der Sortierung einer Spalte wieder. Als Wert liefern diese Eigenschaften jeweils „True" oder „False" zurück.

- **IsCategory**
  Gibt an, ob die Spalte zu Kategorien zusammengefasst wird.
  integerKategorie = NotesViewColumn.IsCategory

- **IsField**
  Enthält die Spalte des Inhalt eines Feldes?
  integerFeld = NotesViewColumn.IsField

- **IsFormula**
  Wird die Spaltenanzeige über eine Formel berechnet?
  integerFormel = NotesViewColumn.IsFormula

- **IsHidden**
  Wird die Spalte nicht angezeigt?
  integerVerborgen = NotesViewColumn.IsHidden

- **IsHideDetail**
  Gibt an, ob Zwischensummen verborgen werden, wenn in der Ansicht eine Summenspalte angezeigt wird.
  integerZwischensumme = NotesViewColumn.IsHideDetail

- **IsIcon**
  Wird der Spalteninhalt als Symbol angezeigt?
  integerIcon = NotesViewColumn.IsIcon

- **IsResize**
  Kann die Spaltenbreite verändert werden?
  integerAendern = NotesViewColumn.IsResize

- **IsResponse**
  Enthält die Spalte nur Antwortdokumente?
  integerAnwort = NotesViewColumn.IsResponse

- **IsShowTwistie**
  Wird das Symbol für das Zusammenfassen bzw. Auseinanderziehen der Ansichtszeilen angezeigt?
  integerSymbol = NotesViewColumn.IsShowTwistie

- **ItemName**
  Gibt den Namen des Feldes an, dessen Inhalt die Spalte anzeigt. Wird der Inhalt der Spalte berechnet, wird der automatisch generierte Name für den Spalteninhalt angezeigt.
  stringFeld = NotesViewColunm.ItemName

- **ListSep**
  Diese Eigenschaft enthält das Trennzeichen bei einem Feld mit Mehrfachwerten.
  integerTrennung = NotesViewColumn.ListSep

  Für die einzelnen Trennzeichen gelten folgende Konstanten-Namen:
  VC_SEP_SPACE – Leerzeichen – (1)
  VC_SEP_COMMA – Komma – (2)
  VC_SEP_SEMICOLON – Semikolon – (3)
  VC_SEP_NEWLINE – Zeilenschaltung – (4)

- **NumberAttrib, NumberDigits, NumberFormat**
  Das Format von Zahlen wird mit diesen Eigenschaften ermittelt.

- **Position**
  Gibt die Position der Spalte in der Ansicht zurück.
  integerPosition = NotesViewColumn.Position
  Die erste Spalte liefert den Wert 1 zurück (im Gegensatz zu der Eigenschaft „Columns" der NotesView-Klasse, deren Elemente mit 0 beginnen).

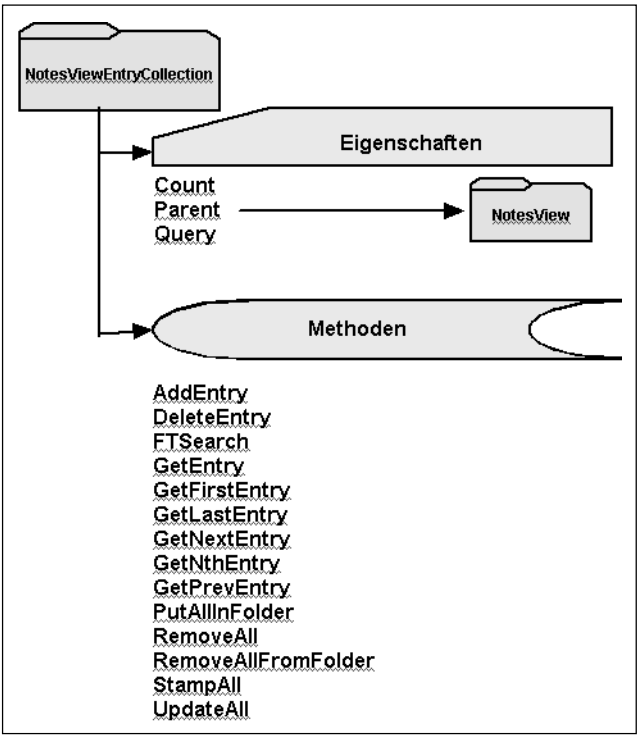
- **TimeDateFmt, TimeFmt, TimeZoneFmt**
  Formate einer Spalte mit Zeitwerten.

- **Title**
  Liefert den Text der Spaltenüberschrift.
  stringTitel = NotesViewColumn.Title

- **Width**
  Spaltenbreite
  intBreite = NoteViewColumn.Width

6.3.6.2.6 Die Klassen NotesViewEntryCollection und NotesViewEntry

Während die Klasse „NotesView" Verweise auf die Dokumente enthält, die der Auswahlformel dieser Ansicht entsprechen, beschäftigt sich die Klasse „NotesViewEntryCollection" mit den Zeilen und ihren Inhalten, die in der Ansicht selbst abgebildet sind. In einer „NotesViewEntryCollection" können alle oder auch nur eine Teilmenge der Zeilen einer Ansicht enthalten sein. Ein Objekt der Klasse „NotesViewEntryCollection" wird über die Eigenschaft „AllEntries" oder die Methode „GetAllEntiesByKey" der Klasse NotesView erzeugt.



Die Klasse NotesViewEntryCollection

Die Eigenschaften der Klasse „NotesViewEntryCollection" sind:

- **Count**
  Die Anzahl der Zeilen, die in der Eintragssammlung enthalten sind.
  longAnzahl = NotesViewEntryCollection.Count
  Diese Eigenschaft ist dann wichtig, wenn geprüft werden soll, ob überhaupt Einträge in dem NotesViewCollection-Objekt verfügbar sind.

- **Parent**
  Die Parent-Eigenschaft enthält die Ansicht, von der die NotesViewEntryCollection abgeleitet wurde in Form eines NotesView-Objektes.
  Set NotesView = NotesViewEntryCollection.Parent

- **Query**
  In der Query-Eigenschaft ist die Filterformel hinterlegt, die bei Erstellung des Objektes verwendet wurde.
  stringFilter = NotesViewEntryCollection.Query

Die folgenden Methoden der Klasse „NotesViewEntryCollection" beziehen sich im Wesentlichen auf Verwaltung von Ansichtszeilen:

- **AddEntry**
  Fügt eine Zeile aus der Ansicht zu der NotesViewEntryCollection hinzu.
  Call _
  NotesViewEntryCollection.AddEntry(NotesViewEntry)

- **DeleteEntry**
  Löscht einen Eintrag aus der NotesViewEntryCollection.
  Call _
  NotesViewEntryCollection.DeleteEntry(NotesViewEntry)

- **FTSearch**
  Die Volltextsuche filtert die Einträge der NotesViewEntryCollection über Suchbegriffe und reduziert die Einträge auf solche, in denen die Suchbegriffe enthalten sind. Die Methode setzt einen Volltextindex in der Datenbank voraus. Ohne Volltextindex in der Datenbank ist das Ergebnis schlechter als mit Volltextindex.
  Call _
  NotesViewEntryCollection.FTSearch(stringSuchbegriff, _
  integerMaxAnzahlDokumente)

- **GetEntry, GetFirstEntry, GetLastEntry, GetNthEntry, GetPrevEntry**
  Diese Methoden dienen der Erstellung eines Objektes der Klasse „NotesViewEntry". Das neue Objekt kann über seine „Document"-Eigenschaft zur Erstellung des entsprechenden NotesDocumentObjektes eingesetzt werden.
  Set NotesViewEntry = _
  NotesViewEntryCollection.GetNthEntry(5)

- **PutAllInFolder**
  Die PutAllInFolder-Methode fügt die Dokumente der Einträge in der NotesViewEntryCollection zu einem Ordner hinzu.
  Call _
  NotesViewEntryCollection.PutAllInFolder(stringOrdner, _
  integerAnlegen)

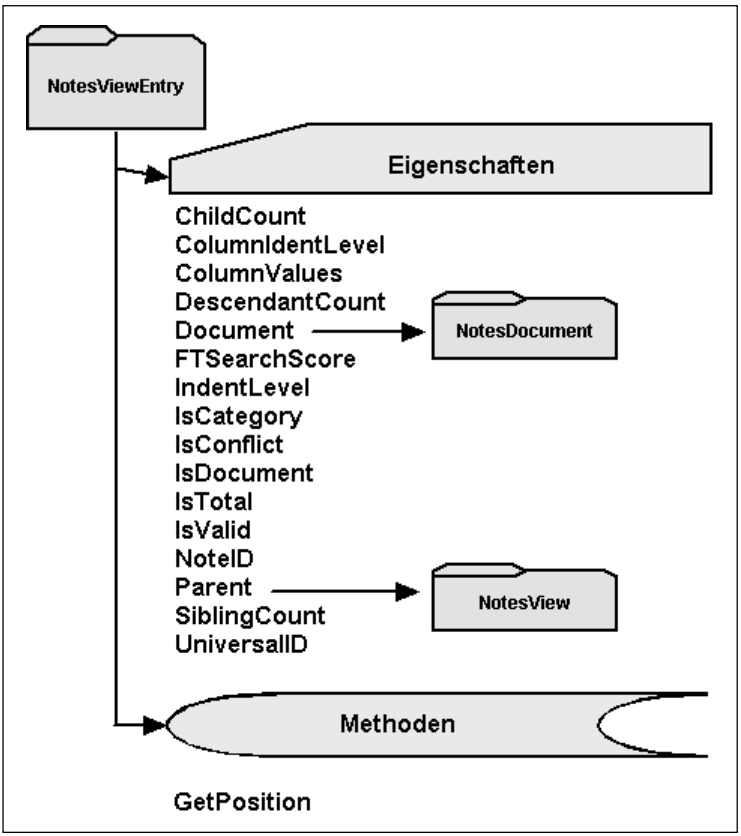
- **RemoveAll**
  löscht alle Dokumente, die in der NotesViewEntryCollection eingetragen sind.
  Call _
  NotexViewEntryCollection.RemoveAll(integerSofort)

- **RemoveAllFromFolder**
  Die RemoveAllFromFolder-Methode löscht die Dokumente der Einträge in der NotesViewEntryCollection aus einem Ordner.
  Call _
  *NotesViewEntryCollection.RemoveAllFromFolder_*
  (stringOrdner)

- **StampAll**
  Mit Hilfe der Methode "StampAll" kann ein Feld in jedem Dokument, das in der NotesViewEntryCollection eingetragen ist, geändert oder, wenn es nicht vorhanden ist, angelegt werden.
  Call NotesViewEntryCollection.StampAll("stringFeld", _
  Wert)

- **UpdateAll**
  markiert alle Dokumente mit einem Eintrag in einer NotesViewEntryCollection als „Bearbeitet durch einen Agenten".
  Call NotesViewEntryCollection.UpdateAll

Die Klasse „NotesViewEntry" bietet Eigenschaften und Methoden zu den einzelnen Einträgen einer NotesViewEntryCollection. Ein Eintrag entspricht einer Zeile in der Ansicht und beinhaltet alle Spalteninformationen zu dieser Zeile.



Die Klasse NoteViewEntry

Ein Objekt der Klasse „NotesViewEntry" kann auf unterschiedliche Art erstellt werden:

- aus einer Ansicht
  Set NotesViewEntry = NotesView.GetEntryByKey(„Hugo")
- aus einer NotesViewEntryCollection
  Set NotesViewEntry = _
  NotesViewEntryCollection.GetFirstEntry
- aus einem NotesViewNavigator
  Set NotesViewEntry = NotesViewNavigator.GetEntry

Die Eigenschaften der Klasse „NotesViewEntry" sind:

- **ChildCount**
  ermittelt die Anzahl der direkten Antwortdokumente zu dem Dokument des Eintrags.
  intergerAnzahl = NotesViewEntry.ChildCount
- **ColumnIndentLevel**
  gibt an, auf welcher Abhängigkeitsebene sich der Eintrag befindet. Handelt es sich bei dem Eintrag um ein Hauptdokument, so liefert ColumnIndentLevel den Wert 0, die erste Antwort liefert den Wert 1 usw.
  integerEbene = NotesViewEntry.ColumnIndentLevel
- **ColumnValues**
  Die Eigenschaft „ColumnValues" enthält die Spaltenwerte eines Eintrages in Form eines Arrays vom Typ Variant. Die Werte einer einzelnen Spalte werden aus dem entsprechenden Element des Arrays entnommen. Der Wert der ersten Spalte befindet sich in ColumnValues(0).
  variantWert = NotesViewEntry.ColumnValues(3)

- **DescendantCount**
  Diese Eigenschaft enthält die Anzahl der Unterdokumente zu einem Dokument und zwar unabhängig von ihrer Abhängigkeitsebene.
  integerAnzahl = NotesViewEntry.DescendantCount

- **Document**
  Das Dokument zu dem aktuellen Eintrag findet sich als Objekt unter der Eigenschaft „Document". Dieses Objekt ist dann erforderlich, wenn auf Felder zugegriffen werden soll, die in den Ansichtsspalten nicht enthalten sind.
  Set NotesDocument = NotesViewEntry.Document

- **FTSearchScore**
  liefert eine Zahl mit dem Ergebnis der Volltextsuche, die zum Auffinden des Eintrags geführt hat.
  integerAnzahl = NotesViewEntry.FTSearchScore

- **IndentLevel**
  gibt die Abhängigkeitsebene des Eintrags zurück.
  integerEbene = NotesView.IndentLevel

- **IsCategory, IsConflict, IsDocument, IsTotal**
  ermöglichen die Abfrage auf Eigenschaften des angegebenen Eintrages. IsDocument z.B. besagt, dass es sich um einen Eintrag zu einem Dokument handelt.
  integerWert = NotesViewEntry.IsTotal

- **IsValid**
  Wenn ein Dokument im Backend gelöscht wurde, in der Ansicht aber noch enthalten ist, liefert „IsValid" den Wert „False" zurück.
  integerWert = NotesViewEntry.IsValid

- **NoteID**
  enthält die NoteID des Eintrages mit einem davor gestellten „NT".
  stringNoteID = NotesViewEntry.NoteID

- **Parent**
  Die „Parent"-Eigenschaft liefert das Ausgangs-Objekt für das NotesViewEntry-Objekt zurück..
  Set NotesView = NotesViewEntry.Parent

- **SiblingCount**
  enthält die Anzahl der Dokumente auf der gleichen Abhängigkeitsebene.
  integerAnzahl = NotesViewEntry.SiblingCount

- **UniversalID**
  Die Eigenschaft „UniversalID" enthält eine 32-stellige Zeichenfolge, die das zum Eintrag gehörige Dokument eindeutig identifiziert.
  stringID = NotesViewEntry.UniversalID

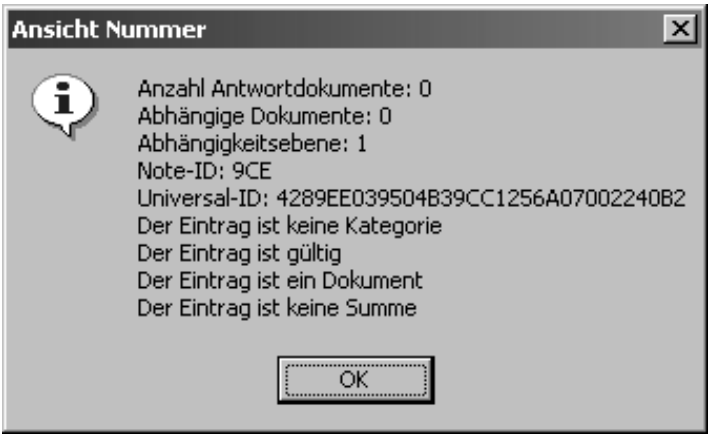
Programmtext

```
'/// Die Funktion zeigt mit Hilfe einer Schaltfläche verschiedene
'/// Eigenschaften eines Eintrages in einer Adressen-Ansicht.

Sub Click
'///
    Dim ses As New NotesSession
    Dim vw As NotesView
    Dim ve As NotesViewEntry
    Dim doc As NotesDocument
```

```
Set vw = ses.CurrentDatabase.GetView("vwAdrNum)
Set ve = vw.GetEntryByKey(InputBox$("Adressnummer:"))
Set doc = ve.Document

strMsg = "Anzahl Antwortdokumente: " & ve.ChildCount & _
  Chr$(10) & "Abhängige Dokumente: "
  & ve.DescendantCount & Chr$(10) _
  & "Abhängigkeitsebene: " & ve.IndentLevel & _
  Chr$(10) & "Note-ID: " & ve.NoteID & Chr$(10) _
  & "Universal-ID: " & ve.UniversalID & Chr$(10)

If ve.IsCategory Then
  strMsg = strMsg & _
       "Der Eintrag ist eine Kategorie" & Chr$(10)
Else
  strMsg = strMsg & _
       "Der Eintrag ist keine Kategorie" & Chr$(10)
End If

If ve.IsValid Then
  strMsg = strMsg & "Der Eintrag ist gültig" & Chr$(10)
Else
  strMsg = strMsg & "Der Eintrag ist ungültig" & _
       Chr$(10)
End If

If ve.IsDocument Then
  strMsg = strMsg & _
       "Der Eintrag ist ein Dokument" & Chr$(10)
Else
  strMsg = strMsg & "Der Eintrag ist kein Dokument" _
       & Chr$(10)
End If
```

```
    If ve.IsTotal Then
      strMsg = strMsg & "Der Eintrag ist eine Summe"
    Else
      strMsg = strMsg & "Der Eintrag ist keine Summe"
    End If

    '/// Eigenschaften in einem Nachrichtenfeld anzeigen
    MsgBox strMsg, 64, "Ansicht " & vw.Name

'///
End Sub
```

Die Nachricht hat folgenden Inhalt:



Eigenschaften eines NotesViewEntry-Objektes

Nur eine Methode steht für die NotesViewEntry-Klasse zur Verfügung:

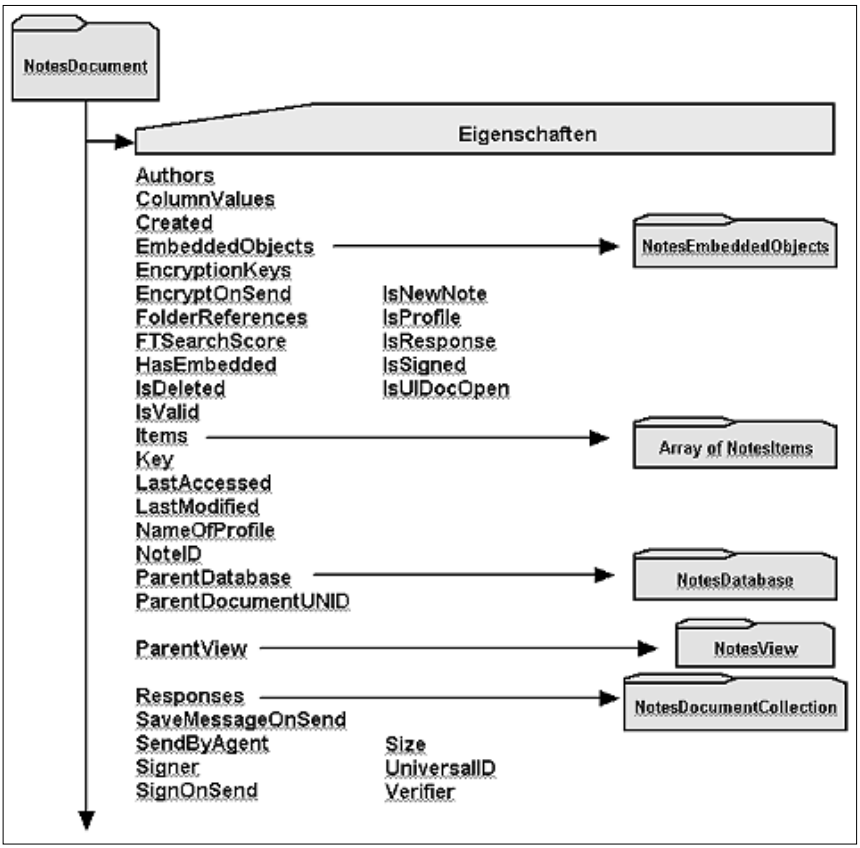
- **GetPosition**
  liefert die Position des Eintrages in der Hierarchie der Ansicht. Die Ebenen sind durch das angegebene Zeichen getrennt. Die Eigenschaft hat den Typ String.
  stringPosition = _
  NotesViewEntry.GetPosition(stringTrennzeichen)

6.3.6.2.7 Die Klasse NotesDocument

Die Klasse NotesDocument beschäftigt sich mit den Dokumenten einer Notes-Datenbank, einer Notes-Ansicht oder einer Notes-Dokumentensammlung. Die Notes-Dokumente entsprechen den Datensätzen in der Welt der relationalen Datenbank. Damit wird deutlich, wie wichtig diese Klasse für die Verarbeitung von Daten einer Notes-Datenbank ist. Entsprechend umfangreich sind auch die angebotenen Eigenschaften und Methoden.

Ein Objekt der Klasse NotesDocument kann auf unterschiedliche Arten erstellt werden. Zum einen bietet sich die Methode „New“ an, zum anderen kann auf ein Objekt der Klasse „NotesDatabase“ zurückgegriffen und dessen Methode „createDocument“ verwendet werden. Dokumente stellen also Datensammlungen dar, die in einer Datenbank abgelegt sind. Die Daten sind dabei zunächst einmal unabhängig von den in der Datenbank verfügbaren Masken. Es können also auch unterschiedliche Masken zur Verarbeitung von Daten eines Dokumentes eingesetzt werden. Ungewöhnlich ist auch, dass die Felder eines Dokumentes von denen anderer Dokumente abweichen können.

Bei den Eigenschaften eine NotesDocument-Objektes bietet sich folgendes Bild:



Eigenschaften der Klasse NotesDocument

Die Eigenschaften beziehen sich größtenteils auf das Dokument selbst, teilweise aber auch auf verbundene Objekte – wie z.B. die übergeordnete Ansicht:

- **Authors**
  Diese Eigenschaft enthält die Namen der Personen, die ein Dokument gespeichert haben. Diese Eigenschaft hat nichts mit den Rechten an dem Dokument zu tun. Die Rechte werden über die ACL (Zugriffskontrollliste) sowie die Felder vom Typ „Autoren“ gesteuert. Die Eigenschaft kann nur gelesen werden.
  stringArray = NotesDocument.Authors

- **ColumnValues**
  Damit die Eigenschaft „ColumnValues“ Werte enthält, muss das Dokument Felder enthalten, die in einer Ansicht aufgeführt sind. Der Spalteninhalt der Ansicht, in der das Dokument erscheint, ist als Array von Variant-Variablen in der Eigenschaft abrufbar. Die Eigenschaft kann nur gelesen werden.
  variantArray = NotesDocument.ColumnValues

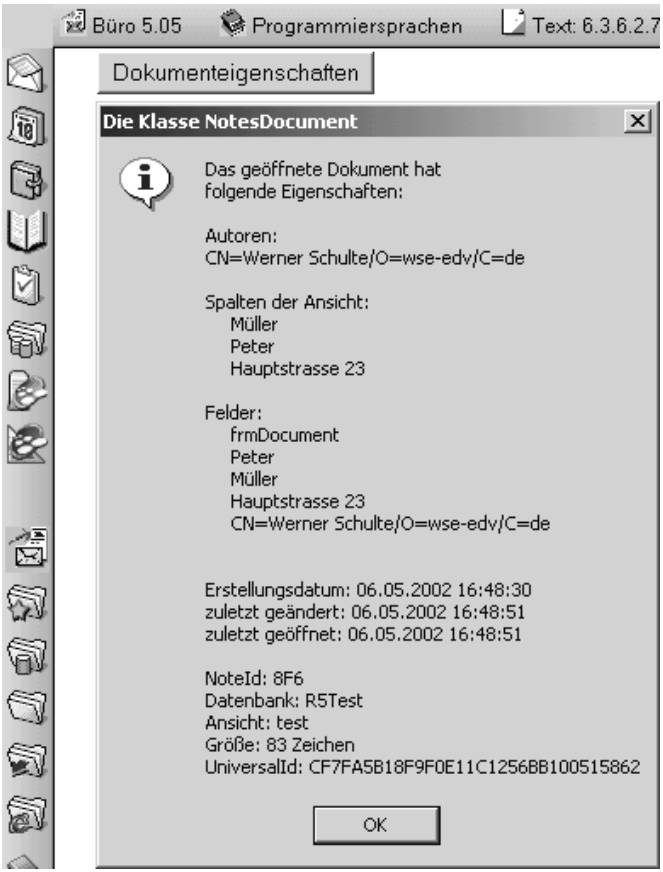
- **Created**
  Liefert das Datum, an dem das Dokument erstellt wurde. Das Ergebnis ist eine Variant-Variable mit einem Wert vom Typ „Datum“.
  variantDatum = NotesDocument.Created

Programmtext

```
Sub ShowProperties
,///
    ,/// NotesSession-Objekt erstellen
    Dim ses As New NotesSession

    '/// NotesView-Objekt definieren
    Dim vw As NotesView

    '/// NotesDocument-Objekt definieren
    Dim doc As NotesDocument

    ,/// Aus der geöffneten Datenbank dem NotesView-Objekt
    ,/// eine Ansicht zuweisen
    Set vw = ses.CurrentDatabase.getView("vwTest")

    '/// Das erste Dokument der Ansicht
    ,/// dem Dokumentobjekt zuweisen
    Set doc = vw.GetFirstDocument

    '/// Dokumenteigenschaften ermitteln
    ,/// und als Msgbox anzeigen
    With doc

      strMsg = "Das geöffnete Dokument hat" _
      & Chr$(10) & "folgende Eigenschaften:" & _
      Chr$(10) & Chr$(10) &_
      "Autoren: " & Chr$(10)

      ,/// Werte des Autorenfeldes ermitteln
      For i = 0 To Ubound(.Authors)
          strMsg = strMsg & .Authors(i) & _
          Chr$(10)
      Next
```

```
strMsg = strMsg & Chr$(10) & _
     “Spalten der Ansicht: “  & Chr$(10)

,/// Die Eigenschaft ColumnValues
Forall c In .ColumnValues
     strMsg = strMsg & Space(5) & c & _
     Chr$(10)
End Forall

,/// Inhalte des Dokumentenfelder übernehmen
strMsg = strMsg & Chr$(10) & “Felder:”  & _
     Chr$(10)

For i = 0 To Ubound(.items)
     strMsg = strMsg & Space(5) & _
     .items(i).Text & Chr$(10)
Next

,/// Datums- und weitere Werte ermitteln
strMsg = strMsg & Chr$(10) & Chr$(10) & _
“Erstellungsdatum: “ & .Created & _
Chr$(10) & _
“zuletzt geändert: “ & .LastModified & _
Chr$(10) & _
“zuletzt geöffnet: “ & .LastAccessed & _
Chr$(10) & Chr$(10) &_
“NoteId: “ & .NoteId & Chr$(10) & _
“Datenbank: “ & .ParentDatabase.Title & _
Chr$(10) & _
“Ansicht: “ & .ParentView.Name & _
Chr$(10) & _
“Größe: “ & .Size  & “ Zeichen “ & _
Chr$(10) & _
“UniversalId: “ & .UniversalId & Chr$(10)
```

```
    ,/// Informationen am Bildschirm ausgeben
    Msgbox strMsg, 64, _
    "Die Klasse NotesDocument"

  End With

End Sub
```



Dokument-eigenschaften anzeigen

- **EmbeddedObjects**
  Unter Windows ist Notes in der Lage, OLE-Objekte in Richtext-Feldern zu speichern. Über die Eigenschaft „EmbeddedObjects“ kann auf diese Objekte zugegriffen werden. Das Ergebnis aus der Abfrage der Eigenschaft ist eine Array mit Objekten der Klasse „NotesEbeddedObject“.
  arrayNotesEmbeddedObject = _
  NotesDocument.EmbeddedObjekts

- **EncryptionKeys**
  In der Eigenschaft „EncryptionKeys“ kann ein Schlüssel hinterlegt werden, mit dem die Methode „Encrypt“ das Dokument verschlüsseln soll. Die Eigenschaft kann gelesen und geändert werden.
  stringArray = NotesDocument.EncryptionKeys
  NotesDocument.EncryptionKeys = stringText oder
  NotesDocument.EncyptionKeys = stringArray

- **EncryptOnSend**
  Ob ein Dokument bei dem Versand als e-Mail verschlüsselt wird oder nicht, wird über die Eigenschaft “EncryptOnSend” festgelegt.
  integerVerschluesseln = NotesDocument.EncryptOnSend
  NotesDocument.EncryptOnSend = integerVerschluesseln

- **FolderReferences**
  Es kann wichtig sein, zu wissen, ob ein Dokument in einem bestimmten Ordner enthalten ist. Damit die Eigenschaft „FolderReferences“ verwendet werden kann, ist zunächst sicherzustellen, dass die Datenbank die beiden verborgenen Ansichten “$FolderRef” und “$FolderRefInfo” enthält. Diese

Ansichten sind in der Schablone der Maildatenbank enthalten und können von dort kopiert werden.
stringArray = NotesDocument.FolderReferences

- **FTSearchScore**
  Bei der Ausführung einer Volltextsuche wird die Anzahl der Treffer für das Dokument in die Eigenschaft „FTSearchScore“ gestellt.
  integerTreffer = NotesDocument.FTSearchScore

- **HasEmbedded**
  Wenn ein Dokument eingebettete OLE-Objekte enthält, wird die Eigenschaft „HasEmbedded“ auf „True“ gesetzt.
  integerAntwort = NotesDocument.HasEmbedded

- **IsDeleted**
  Diese Eigenschaft enthält „True“ bzw. „False“ – je nachdem, ob sie gelöscht wurden oder nicht.
  integerGeloescht = NotesDocument.IsDeleted

- **IsEncrypted (Version 6)**
  liefert zurück, ob das Dokument verschlüsselt ist oder nicht.
  integerVerschluesselt = NotesDocument.IsEncrypted

- **IsNewNote**
  Die Eigenschaft „IsNewNote“ bleibt solange auf „True“, bis das Dokument zum ersten Mal gespeichert wird.
  integerNeu = NotesDocument.IsNewNote

- **IsProfile**
  Profildokumente werden im Prinzip genauso behandelt, wie andere Dokumente. Die Erstellung des Dokument-Objektes für ein Profildokument erfolgt über spezielle Methoden (z.B. NotesDatabase.GetProfileDocument). Sobald das Objekt er-

stellt ist, gelten dafür die Eigenschaften und Methoden der Klasse NotesDocument. Über die Eigenschaft „IsProfile“ kann ermittelt werden, ob es sich um ein Profildokument handelt oder nicht.
integerProfil = NotesDocument.IsProfile

- **IsResponse**
  Diese Eigenschaft enthält „True“, wenn es bei dem Dokument um ein Antwortdokument handelt.
  integerAntwort = NotesDocument.IsResponse

- **IsSigned**
  Wenn das Dokument signiert wurde, enthält es ein Feld vom Typ „Signature“. Die Eigenschaft „IsSigned“ wird dann auf „True“ gesetzt.
  integerSigniert = NotesDocument.IsSigned

- **IsUiDocOpen**
  Objekte für ein Notes-Dokument können aus einem geöffneten Frontend-Dokument (NotesUiDocument) heraus mit Hilfe der Eigenschaft „Document“ erstellt werden. Dann ist sowohl ein Objekt der Klasse „NotesUiDocument“ als auch eines der Klasse „NotesDocument“ verfügbar. Die Eigenschaft „IsUiDocOpen“ enthält den Wert „True“, wenn diese Situation gegeben ist.
  integerOpen = NotesDocument.IsUiDocOpen

- **IsValid**
  Ein gelöschtes Dokument wird mit Hilfe eines Restdokumentes für Replizierungszwecke weitergeführt. Ob es sich um ein solches Restdokument oder ein solches mit gültigen Feldern handelt, kann in dieser Eigenschaft abgelesen werden.
  integerValid = NotesDocument.isValid

- **Items**
  Da der Begriff „Felder“ an Masken gebunden ist, verwendet man zur Kennzeichnung der Dokumentinhalte den Begriff „Item“, um damit dem Umstand Rechnung zu tragen, dass die Notes-Dokumente Inhalte haben können, für die in der dazugehörigen Maske keine Felder zu finden sind. Ein Item ist also ein Datenfeld in einem Dokument. Die Eigenschaft „Items“ liefert die Namen aller Datenfelder eines Dokumentes als Liste zurück.
  itemArray = NotesDocument.items

- **Key**
  Schlüsselbegriff des Dokumentes, wenn es sich um ein Profildokument handelt.
  stringKey = NotesDocument.key

- **LastAccessed**
  Diese Eigenschaft enthält das Datum, an dem zum letzten Mal auf das Dokument zum Lesen oder Schreiben zugegriffen wurde.
  variantDatum = NotesDocument.LastAccessed

- **LastModified**
  Datum, an dem das Dokument zuletzt geändert wurde.
  variantDatum = NotesDocument.LastModified

- **LockHolders (Version 6)**
  Wenn ein Dokument durch den Lock-Befehl für einen oder mehrere Benutzer gesperrt wurde, enthält diese Eigenschaft die Namen der Benutzer, für die das Dokument gerade reserviert ist.
  stringArray = NotesDocument.LockHolders

- **NameOfProfile**
  Wenn es sich bei dem Dokument um ein Profildokument handelt, kann über diese Eigenschaft der Name des Profils ermittelt werden.
  stringName = NotesDocument.NameOfProfile

- **NoteId**
  Notes vergibt für jedes Dokument einer Datenbank einen internen Namen, die Note-Id. Es handelt sich dabei um einen 8-stelligen Begriff aus Buchstaben und Zahlen. Dieser Name wird mit der Eigenschaft „NoteId“ ermittelt.
  stringNoteId = NotesDocument.NoteId

- **ParentDatabase**
  Name der Datenbank, in der das Dokument gespeichert ist.
  NotesDatabase = NotesDocument.ParentDatabase

- **ParentDocumentUNID**
  Diese Eigenschaft ist nur bei Antwortdokumenten verfügbar. Sie enthält die Kennung des Hauptdokumentes, von dem das Antwortdokument abhängt.
  stringId = NotesDocument.ParentDocumentId

- **ParentView**
  Name der Ansicht, die verwendet wurde, um das NotesDocument-Objekt zu erstellen.
  NotesView = NotesDocument.ParentView

- **Responses**
  Alle Antwortdokumente zu einem Dokument als Dokumentensammlung. Antwortdokumente zu Antwortdokumenten werden dabei nicht berücksichtigt.
  NotesDocumentCollection = NotesDocument.Responses

- **SaveMessageOnSend**
  Diese Eigenschaft steuert, ob ein Dokument gespeichert wird, wenn es als Mail versendet wird. Nach dem Versand kann aus ihr abgelesen werden, ob das Dokument beim Versand gespeichert wurde. Die Eigenschaft wird nur bei einem neuen (noch nicht gespeicherten) Dokument gesetzt.
  integerVersandt = NotesDocument.SaveMessageOnSend
  NotesDocument.SaveMessageOnSend = TRUE / FALSE

- **SentByAgent**
  Diese Eigenschaft zeigt an, ob das Dokument durch einen Agenten versendet wurde.
  integerVersandt = NotesDocument.SentByAgent

- **Signer**
  Name des Benutzers, der das Dokument signiert hat (wenn es signiert wurde).
  stringName = NotesDocument.Signer

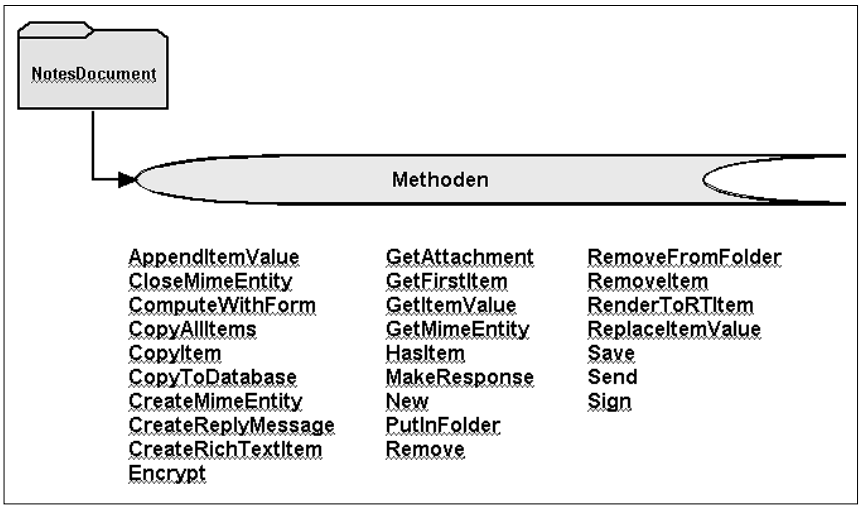
- **SignOnSend**
  Ein Dokument kann, wenn es verschickt wird, mit einer Signatur versehen werden. Die Eigenschaft „SignOnSend“ kann dazu verwendet werden, die Signatur beim Versand auszulösen oder dazu, nach dem Versand festzustellen, ob das Dokument signiert wurde.
  integerSign = NotesDocument.SignOnSend
  NotesDocument.SignOnSend = TRUE / FALSE

- **Size**
  Die Größe eines Dokumentes in Byte (einschließlich aller Dateianhänge) kann über die Size-Eigenschaft ermittelt werden.
  longGröße = NotesDocument.Size

- **UniversalID**
  Im Gegensatz zur NoteID ist „UniversalID“ ein Kennzeichen eines Dokumentes, das es generell nur einmal geben sollte. Die UniversalId hat 32 Stellen. Sie wird dazu verwendet, Dokumente in Datenbankrepliken miteinander abzugleichen.
  Wird die UniversalId eines Dokumentes geändert, wird es zu einem neuen Dokument.
  stringUNId = NotesDocument.UniversalId
  NotesDocument.UniversalId = stringUNId

- **Varifier**
  Name der Zulassungsstelle, die ein Zertifikat ausgestellt hat, mit dem das Dokument signiert wurde.
  stringName = NotesDocument.Verfier

Die NotesDocument-Klasse bietet folgende Methoden an:



Methoden der NotesDocumentklasse

- **AppendItemValue**
  Mit Hilfe der Methode „AppendItemValue“ kann ein neues Datenfeld angelegt sowie mit einem Inhalt und einem Datentyp versehen werden.
  Set NotesItem = NotesDocument.AppendItemValue („Feldname“, „Wert“)

- **CloseMimeEntities, CreateMimeEntity, GetMimeEntity (Version 6)**
  Methoden für die MIME-Verarbeitung eines Feldes.
  integerErgebnis = _
  NotesDocument.CloseMimeEntities(intSpeichern, _
  stringFeldName)
  Set NotesMIMEEntity = _
  NotesDocument.CreateMIMEEntity([stringName])
  Set NotesMIMEEntity = _
  NotesDocument.GetMIMEEntity([stringName])

- **ComputeWithForm**
  Mit Hilfe der ComputeWithForm-Methode werden die einzelnen Felder eines Dokuments überprüft. Dabei werden Vorgabewerte gesetzt, Feldprüfungen durchgeführt und Formeln berechnet.
  integerErgebnis = _
  NotesDocument.ComputeWithForm (intTypKontrolle, _
  integerFehlerRückgabe)

- **CopyAllItems**
  kopiert alle Felder des Dokumentes in ein anderes Dokument.
  Call NotesDocument.CopyAllItems(NotesDocument _
  [, integerErsetzen])

- **CopyItem**
  kopiert ein Feld eines Dokumentes in das aktuelle Dokument. Der Rückgabewert ist ein Objekt mit dem Typ „NotesItem“.
  NotesItem = NotesDocument.CopyItem(NotesItem, _ stringNeuerName)

Programmtext

```
,/// Aus einem bestehenden, geöffneten Dokument wird eine
,/// Aufgabe in der Maildatenbank des Benutzers erstellt.

Sub CreateToDo
‘///
  ‘/// NotesSession-Objekt erstellen
  Dim ses As New NotesSession

  ‘/// NotesUiWorkspace-Objekt erzeugen
  Dim ws As New NotesUIWorkspace

  ‘/// NotesDatabase-Objekt für die Maildatenbank
  ,/// des Benutzers erzeugen
  Dim dbMail As New NotesDatabase(“”,””)

  ‘/// NotesDocument-Objekt für die Aufgabe
  Dim docEntry As NotesDocument

  ‘/// NotesUiDocument-Objekt für das
  ,/// geöffnete Dokument
  Dim uidoc As NotesUIDocument

  ,/// NotesUiDocument-Objekt zur Bearbeitung
  ,/// der neuen Aufgabe
  Dim uidocEntry As NotesUIDocument
```

```
‘/// NotesDocument-Objekt für das geöffnete
Dokument
Dim doc As NotesDocument

,/// Felder der neuen Aufgabe
Dim itmTermin As NotesItem
Dim itmBody As NotesItem

Dim strBetreff As String
Dim strUserMailServer As String
Dim strUserMailDb As String

‘/// NotesUiDocument-Objekt für das geöffnete
,/// Dokument aus dem NotesUiWorkspace-
,/// Objekt erstellen
Set uidoc = ws.CurrentDocument

,/// Server und Maildatenbank des Benutzers
,/// aus der Datei Notes.ini ermitteln
strUserMailServer = _
ses.GetEnvironmentString(“MailServer”, True)
strUserMailDb = _
ses.GetEnvironmentString(“MailFile”, True)

‘/// Betreff-Feld aus dem geöffneten Dokument
,/// übernehmen
strBetreff = uidocCur.FieldGetText(“Betreff”)

,/// Aktuelles Dokument speichern
uidoc.Save

‘/// NotesDocument-Objekt des aktuellen
‘/// Dokumentes erstellen
Set doc = uidoc.Document
```

```
,/// NotesDatabase-Objekt für die Maildatenbank
,/// des Benutzers erstellen
Set dbMail = New _
NotesDatabase(strUserMailServer, _
strUserMailDb)

‘/// Neues Dokument in der Maildatenbank
,/// erstellen
Set docEntry = dbMail.CreateDocument

‘/// Das Aufgaben-Formular dem neuen
,/// Dokument zuweisen
docEntry.Form = “Task”

,/// Felder füllen
docEntry.Subject = strBetreff
docEntry.Importance = “2”
docEntry.chair = ses.UserName
docEntry.AssignState = 0

,/// Terminfelder mit Hilfe der Methode
,/// CopyItem übernehmen
Set itmTermin = doc.GetFirstItem(“datRemind”)
Call docEntry.CopyItem(itmTermin, _
“StartDate”)
Set itmTermin = doc.GetFirstItem(“datEnd”)
Call docEntry.CopyItem(itmTermin, “DueDate”)

‘/// Textfeld übernehmen
Set itmBody = doc.GetFirstItem(“Body”)
Call docEntry.CopyItem(itmBody, “Body”)
```

```
    '/// Neues Dokument zum Editieren öffnen
    Set uidocEntry = ws.EditDocument(True, _
    docEntry)
'///
End Sub
```

- **CopyToDatabase**
  Wenn eine Kopie des aktuellen Dokumentes in einer anderen Datenbank erstellt werden soll, so wird die Methode „CopyToDatabase" eingesetzt.
  Set NotesDocument = _
  NotesDocument.CopyToDatabase(NotesDatabase)

- **CreateReplyMessage**
  Mit Hilfe dieser Methode kann ein Dokument für die Beantwortung einer e-Mail erstellt werden. Der Parameter „integerAlle" entscheidet, ob die Antwort an alle Empfänger des Originaldokumentes oder nur an den Absender geschickt wird.
  Set NotesDocument = _
  NotesDocument.CreateReplyMessage(integerAlle)

- **CreateRichTextItem**
  dient der Erstellung eines Richtext-Feldes.
  Set NotesRichTextItem = _
  NotesDocument.CreateRichTextItem(stringName)

Programmtext

```
,/// Datenbank öffnen, Dokument suchen und
,/// erstellen, wenn es nicht gefunden wurde.

  Sub setBook(strBuch As String)
  '///
    ,/// NotesSession-Objekt erstellen
    Dim ses As New NotesSession

    '/// NotesDatabase-Objekt erstellen
    Dim db As NotesDatabase

    '/// NotesView-Objekt definieren
    Dim vw As NotesView

    '/// NotesDocument-Objekt definieren
    Dim doc As NotesDocument

    ,/// Aktuelle Datenbank ermitteln
    Set db = ses.CurrentDatabase

    '/// NotesView-Objekt erstellen
    Set vw = db.GetView("vwkBuch")

    '/// Es geht auch ohne Datenbank-Objekt,
    ,/// weil die aktuelle Datenbank im Session-
    ,/// Objekt enthalten ist:
    ,/// Set vw = _
    '/// ses.CurrentDatabase.GetView("vwkBuch")

    '/// NotesDocument-Objekt erstellen
    Set doc = vw.GetDocumentByKey(strBuch)
```

```
        '/// Prüfen, ob ein NotesDocument-Objekt
        ,/// erstellt wurde
        If doc Is Nothing Then

            '/// Wenn kein Dokument-Objekt erstellt
            ,/// wurde, neues Dokument erstellen
            Set doc = _
            ses.CurrentDatabase.CreateDocument

            With doc

                ,/// Felder füllen
                .form = "frmkBuch"
                .kBuch = strBuch

                ,/// Dokument speichern
                .save True, False
            End With

        End If

    '///
    End Sub

,/// Dateien aus einem Verzeichnis an ein
,/// vorhandenes oder neues Dokument anhängen

    Sub attachFiles
    '///
        '/// NotesSession-Objekt erstellen
        Dim ses as New NotesSession

        '/// NotesUiWorkspace-Objekt erstellen
        Dim ws as New NotesUiWorkspace
```

```
        '/// NotesDocumentCollection-Objekt
        '/// definieren
        Dim col as NotesDocumentCollection

        '/// NotesDocument-Objekt definieren
        Dim doc as NotesDocument

        '/// NitesRichtextItem definieren
        Dim rtItem As Variant

        ,/// NotesEbbededObject für den Datei-Anhang
        ,///definieren
Dim objAtt As NotesEmbeddedObject

Dim strFile as String
Dim strPath as string

        '/// Feststellen, ob der Benutzer Dokumente
        ,/// markiert hat.
        ,/// Wenn ja: Erstes Dokument verwenden
        ,/// Wenn nein: Neues Dokument erstellen


        ,/// NotesDocumentCollection aus der
        ,/// aktuellen Datenbank erstellen
        Set col = _
        ses.CurrentDatabase.UnprocessedDocuments

,/// Wenn der Zähler-Index der
,/// Dokumentensammlung gleich 0
,/// ist, wurde kein Dokument markiert
If col.Count = 0 Then

        '/// Dokument erstellen
        ,/// Da das Session-Objekt ein Objekt
```

```
,/// für die aktuelle Datenbank
,/// enthält, kann direkt auf dessen
,/// CreateDocument-Methode
,/// zugegriffen werden
Set doc = _
ses.CurrentDatabase.CreateDocument

'/// Felder füllen
With doc
        .Form = "frmDokument"
        .Betreff = "Datei-Anhänge"
        .FormularTyp = "Leerblatt"
        .anoDocAutor = ses.UserName
        .FormularName = "Handbuch"
        .strStatus = "Offen"
        .DokArt = "Text"
        .datWriteDay = Today
        .datEnd = Today
        .Bearbeiter = _
                ses.CommonUserName
        .FolgeAktion = "Bearbeiten"
        .Buch = " Datei-Anhänge "
        .SachGebiet = ""
        .SaNr = "001"
        .Kapitel = ""
        .KapNr = "001"
        .PageNr = "000"
        .TextAutor = _
        ses.CommonUserName
        .TitelText = "Text"
        .save True, False
End With
```

```
    Else
        ,/// Ausgewähltes Dokument verwenden
        Set doc = col.getFirstDocument
    End If

    ,/// Ausgangsverzeichnis definieren
    strPath = "C:\Temp\"

    '/// Erste Datei ermitteln
    strFile = Dir$(strPath, 0)

        Do while strFile <> ""
            strFile = strPath & strFile

            ,/// Objekt für das Body-Feld
            ,/// erstellen
        Set rtitem = _
        doc.createRichTextItem("Body")

        ,/// Objekt für den Datei-Anhang
        ,/// erstellen
        Set objAtt = _
        rtitem.embedObject(EMBED_ATTACHMENT, _
        "", strFile)

        ,/// Dokument speichern
        doc.save True, False

        '/// Neuen Datei-Anhang ermitteln
        strFile = Dir$()
    loop
    '///
End Sub
```

```
,/// Statusfeld in ausgewählten Dokumenten ändern

    Sub changeStatus(strNewStatus As String)
    '///
      Dim ws As New NotesUiWorkspace
      Dim ses As New NotesSession
      Dim col As NotesDocumentCollection
      Dim doc As NotesDocument
      Dim i As Integer

      '/// Markierte Dokumente in eine Dokumenten-
      ,/// Sammlung übernehmen
      Set col = _
      ses.CurrentDatabase.UnprocessedDocuments

      ,/// Im Rahmen einer For-Schleife alle
      ,/// Dokumente öffnen
      ,/// und das Feld strStatus mit dem als
      ,/// Parameter übergebenen Status füllen.

      For i = 1 To col.Count

          '/// NotesDocument-Objekt aus dem i-ten
          ,/// Element der Dokument-Sammlung
          ,/// wählen
          Set doc = col.GetNthDocument( i )

          '/// Feld ändern
          doc.strStatus = strNewStatus

          ,/// Dokument speichern
          doc.Save True, False
      Next i
```

```
    ,/// Anzeige der Ansicht aktualisieren
    ws.ViewRefresh
  '///
  End Sub
```

- **Encrypt**
  Diese Methode wird dazu verwendet, das Dokument zu verschlüsseln. Die Verschlüsselung erfolgt zusammen mit dem Speichern des Dokumentes.
  Call NotesDocument.Encrypt

- **GetAttachment**
  Zur Bearbeitung eines Dateianhanges wird dieser mit Hilfe der Methode „GetAttachment“ in ein Objekt der Klasse „NotesEmbeddedObject“ umgewandelt.
  Set NotesEmbeddedObject = _
  NotesDocument.GetAttachment(stringDateiName)

- **GetFirstItem**
  Innerhalb eines Dokumentes kann mit Hilfe der Methode „GetFirstItem“ ein Objekt der Klasse NotesItem für ein Datenfeld erstellt werden.
  Set NotesItem = _
  NotesDocument.GetFirstItem(stringName)

- **GetItemValue**
  Soll nicht ein Item-Objekt erstellt, sondern der Inhalt des Datenfeldes ermittelt werden, so liefert die Methode „GetItemValue“ ein Array mit den Werten des angegebenen Feldes.
  arrayDaten = _
  NotesDocument.GetItemValue(stringName)

- **HasItem**
  gibt zurück, ob ein bestimmtes Datenfeld im Dokument enthalten ist.
  integerErgebnis = NotesDocument.HasItem(stringName)

- **Lock (Version 6)**
  sperrt ein Dokument für den Zugriff durch andere Benutzer. Damit wird verhindert, dass ein Dokument durch verschiedene Benutzer geändert wird. Die Methode wirkt nur, wenn auf die Server-Datenbanken zugegriffen wird.
  integerErgebnis = _
  NotesDocument.Lock([stringName] [, _
  integerProvisorischeSperrung])

- **LockProvisional (Version 6)**
  ermöglicht eine vorläufige Sperrung eines Dokumentes für einen Benutzer oder eine Benutzergruppe.
  integerErgebnis = _
  NotesDocument.LockPrivisional([stringName])

- **MakeResponse**
  macht ein Dokument zum Antwortdokument eines anderen Dokumentes.
  Call _
  NotesDocument_A.MakeResponse(NotesDokument_B)

- **New**
  Mit Hilfe der „New"-Mothode kann ein Objekt der Klasse „NotesDocument" erstellt werden.

- **PutInFolder**
  fügt das Notes-Dokument zu den Dokumenten eines Ordners hinzu. Ist der Ordner in der Datenbank nicht vorhanden, kann der Ordner angelegt werden.
  Call NotesDocument.PutInFolder(stringOrdner [, _ integerAnlegen])

- **Remove**
  löscht ein Dokument aus der Datenbank. Das Dokument kann wieder hergestellt werden, wenn die Datenbankeigenschaft „Wiederherstellbare Löschungen zulassen" eingeschaltet ist.
  integerErgebnis = _ NotesDocument.Remove(integerSofort)

- **RemoveFromFolder**
  entfernt ein Dokument aus einem Ordner. Es bleibt weiter in der Datenbank bestehen.
  Call NotesDocument.RemoveFromFolder(stringOrdner)

- **RemoveItem**
  löscht ein Datenfeld aus einem Dokument. Sind mehrere Datenfelder mit dem angegebenen Namen in dem Dokument enthalten, werden alle Datenfelder mit diesem Namen gelöscht.
  Call NotesDocument.RemoveItem(stringName)

- **RemovePermanently**
  löscht ein Dokument aus der Datenbank, auch wenn die Datenbankeigenschaft „Wiederherstellbare Löschungen zulassen“ eingeschaltet ist.
  integerErgebnis =
  NotesDocument.RemovePermanently(integerSofort)

- **RenderToRTItem**
  Mit Hilfe der Methode „RenderToRTItem“ wird der Inhalt des Dokumentes zusammen mit der Maske zu einem Bild umgewandelt und in dem angegebenen Richtext-Feld gespeichert. Formeln für berechnete Felder und Feldprüfungen werden vor der Erstellung des Bildes durchgeführt.
  integerErgebnis =
  NotesDocument.RenderToRTItem(NotesRichTextItem)

- **ReplaceItemValue**
  ändert den Inhalt eines Datenfeldes des Dokumentes. Ist das Datenfeld nicht vorhanden, wird es angelegt.
  Set NotesItem = _
  notesDocument.ReplaceItemValue(stringItemName)

- **Save**
  speichert das Dokument auf den Datenträger.
  integerErgebnis = NotesDocument.Save(integerSofort, _
  integerAntwort [, integerGelesen])

- **Send**
  versendet das Dokument als e-Mail. Für die Adressierung werden die Standardfelder „SendTo“, „CopyTo“ und „BlindCopyTo“ verwendet.
  Call NotesDocument.Send(integerMaske [, _
  stringArrayEmpfänger])

Programmtext

```
,/// Aktuelles Dokument als Memo der Maildatenbank aufbereiten
,/// und versenden.

,/// Wenn das aktuelle Dokument nicht alle erforderlichen Felder
,/// enthält, müssen die Daten auf anderem Weg (z.B. über eine
,/// Dialogbox ermittelt werden.

  Sub mailDocument
  '///
    Dim ses As New NotesSession
    Dim ws As New NotesUIWorkspace
    Dim dbMail As New NotesDatabase("", "")
    Dim docMail As NotesDocument
    Dim uidoc As NotesUIDocument
    Dim doc As NotesDocument
    Dim item As NotesItem

    '/// Aktuelles Dokument ermitteln und
    ,/// NotesUiDocument-Objekt erstellen
    Set uidoc = ws.CurrentDocument

    '/// Wenn das geöffnete Dokument im
    ,/// Bearbeitungs-Modus ist, sichern,
    ,/// damit das Dokument auf der
    ,/// Festplatte den gleichen Inhalt hat.
    If uidoc.EditMode = True Then uidoc.Save

    '/// Maildatenbank des Benutzers öffnen
    dbMail.OpenMail
```

```
‘/// NotesDocument-Objekt des aktuellen
‘/// Dokumentes erstellen
Set doc = uidoc.Document

,/// Memo-Dokument in der Maildatenbank des
,/// Benutzers erstellen
Set docMail = dbMail.CreateDocument

,/// Memo-Maske zuweisen
docMail.form = “Memo”

,/// Felder kopieren
,/// Das Feld MailAdress des aktuellen
,/// Dokumentes enthält in diesem
,/// Fall die e-Mail-Adresse des
,/// Empfängers
Set item = doc.GetFirstItem(“MailAdress”)
Call docMail.CopyItem(item, “SendTo”)
Set item = doc.GetFirstItem(“Subject”)
Call docMail.CopyItem(item, “Subject”)
Set item = doc.GetFirstItem(“Body”)
Call docMail.CopyItem(item, “Body”)

‘/// Das Feld From des aktuellen
,/// Dokumentes enthält die Mail-Adresse
,/// des Absenders
Set item = doc.GetFirstItem(“From”)
Call docMail.CopyItem(item, “From”)

‘/// Die Felder zu den Mail-Optionen müssen
,/// bereits gefüllt sein
Set item = doc.GetFirstItem(“DeliveryReport”)
Call docMail.CopyItem(item, “DeliveryReport”)
```

```
    Set item = doc.GetFirstItem("DeliveryPriority")
    Call docMail.CopyItem(item, "DeliveryPriority")
    Set item = doc.GetFirstItem("ReturnReceipt")
    Call docMail.CopyItem(item, "ReturnReceipt")
    Set item = doc.GetFirstItem("MailSaveOptions")
    Call docMail.CopyItem(item, _
    "MailSaveOptions")

    ,/// Memo-Dokument versenden
    docMail.Send False
    docMail.PostedDate = Today

    '/// Memo-Dokument speichern
    docMail.Save True, False, True
  '///
  End Sub
```

- **Sign**
  Signiert das Dokument.
  Call NotesDocument.Sign

- **UnLock (Version 6)**
  gibt ein gesperrtes Dokument wieder frei.
  Call NotesDocument.Unlock

Programmtext

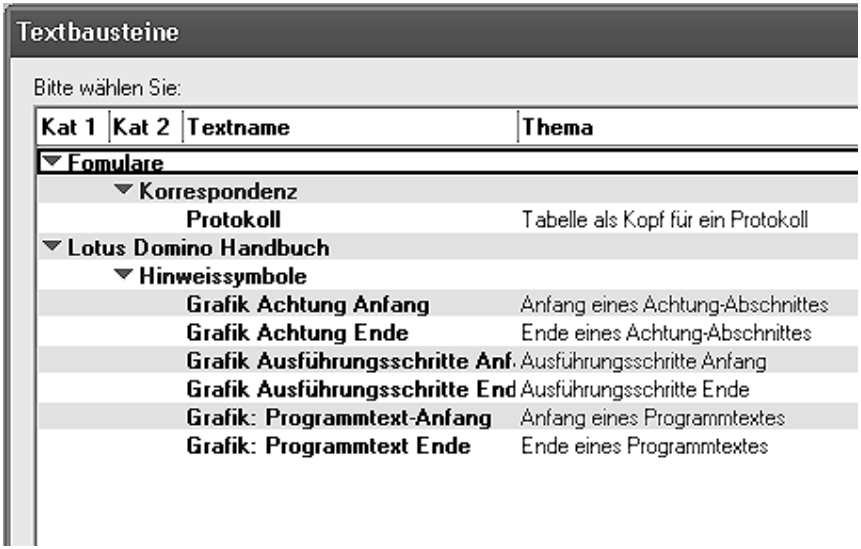
```
,/// Dokument mit Textbaustein auswählen und in das aktuelle
,/// Dokument einfügen

  Sub insertText
  ,///
    Dim ws As New NotesUiWorkspace
    Dim ses As New NotesSession

    '/// NotesUiDocument-Objekt für das
    ,/// Textbaustein-Dokument definieren
    Dim uidocText As NotesUiDocument

    '/// NotesUiDocument-Objekt für das
    '/// aktuelle Dokument definieren
    Dim uidoc As NotesUiDocument

    ,/// NotesDatabase-Objekt für die Textbaustein-
    ,/// Datenbank definieren
    Dim dbText As NotesDatabase

    '/// NotesView-Objekte definieren
    Dim vwText As NotesView
    Dim vw As NotesView

    '/// NotesDocument-Objekt für das Textbaustein-
    ,/// Dokument definieren
    Dim docText As NotesDocument

    Dim varResult As Variant
    Dim strFieldName As String

    Dim item As NotesItem
```

```
‘/// NotesUiDokument-Objekt für das aktuelle
,/// Dokument erstellen
Set uidoc = ws.CurrentDocument

‘/// Aktuelles Dokument in den Bearbeitungs-
,/// Modus setzen, wenn es im Lese-Modus ist
If uidoc.EditMode = False Then _
    uidoc.EditMode = True

,/// Feldname des aktuellen Feldes ermitteln
strFieldName = uidoc.CurrentField

,/// Wenn das Feld nicht Body heißt, dann
,/// die Bearbeitung ablehnen
If strFieldName <> “Body” Then
    Msgbox “Textbausteine können nur” & _
    Chr$(10) & _
    “in das Textfeld eingefügt werden!”, 16, _
    “Achtung”

    ,/// Prozedur beenden
    Exit Sub
End If

‘/// Name der Textbausteindatei angeben
,/// Das kann z.B. über einen Parameter an
,/// die Prozedur übergeben werden
,/// Hier wird zur Vereinfachung eine
,/// String-Konstante eingesetzt
strTextDb = “Textbausteine.nsf”

,/// NotesDatabase-Objekt erstellen
,/// Hier wird der Server der aktuellen
```

```
,/// Datenbank eingesetzt. Es könnte auch ein
,/// anderer Server verwendet werden.
Set dbStart = New _
NotesDatabase(ses.CurrentDatabase.Server, _
      strTextDb)

‘/// Wenn die Textbaustein-Datenbank nicht
,/// geöffnet werden konnte, dann die Prozedur
,/// verlassen
If Not dbText.IsOpen Then Exit Sub

‘/// NotesView-Objekt zur Suche des
,/// Textbausteines erstellen
,/// Der Name der Ansicht muss einer
,/// vorhandenen Ansicht entsprechen
Set vwText = _
dbText.getView(“vwBookTextNam”)
If vwText Is Nothing Then Exit Sub

‘/// Prüfen, ob Textbaustein-Dokumente in der
,/// Ansicht vorhanden sind.
Set docText = vwText.getFirstDocument
If docText Is Nothing Then Exit Sub

‘/// Auswahlliste der vorhandenen Textbausteine
,/// in einem Dialogfeld anzeigen
varResult = _
ws.Pickliststrings(PICKLIST_CUSTOM, _
False, ses.CurrentDatabase.Server, _
strTextDb, “vwBookTextNam”, _
```

```
"Textbausteine", "Bitte wählen Sie:", 5)

,/// Nur weitermachen, wenn ein Ergebnisstring
,/// erzielt wurde
If Isempty(varResult) Then Exit Sub

'/// In der angegebenen Spalte der Ansicht wird
,/// die UniversalDocumentId des
,/// Textbausteindokumentes angezeigt und damit
,/// in die Variable varResult übergeben.
Set vw = dbText.GetView("vwTexteDocId")
Set docText = _
    vw.GetDocumentByKey(varResult(0))

'/// Wenn ein Dokument gefunden wird, das
,/// Dokument im Editier-Modus öffnen
If Not docText Is Nothing Then

    '/// Damit beim Schließen des
    ,/// Textbausteindokumentes die Abfrage
    ,/// zum Speichern unterbleibt, das Feld mit
    ,/// dem Namen SaveOptions auf "0"
    ,/// setzen.
    docText.SaveOptions = "0"

    ,/// NotesUiObjekt für das Textbaustein-
    ,/// Dokument erstellen und das Dokument
    ,/// im Editiermodus öffnen

    ,/// Cursor auf das Feld Body setzen
    uidocText.GotoField "Body"
```

```
        '/// Inhalt des Body-Feldes markieren
        uidocText.SelectAll

        ,/// Inhalt des Body-Feldes in die
        ,/// Zwischenablage (Clipboard)
        ,/// von Windows kopieren
        uidocText.Copy

        ,/// Inhalt des Body-Feldes aus der
        ,/// Zwischenablage an der Cursorposition
        ,/// des aktuellen Dokumentes einfügen
        uidoc.Paste

        ,/// Textbausteindokument schließen
        uidocText.close

        ,/// Formeln des aktuellen Dokumentes
        ,/// aktualisieren
        uidoc.Refreshhideformulas

    End If
'///
End Sub
```

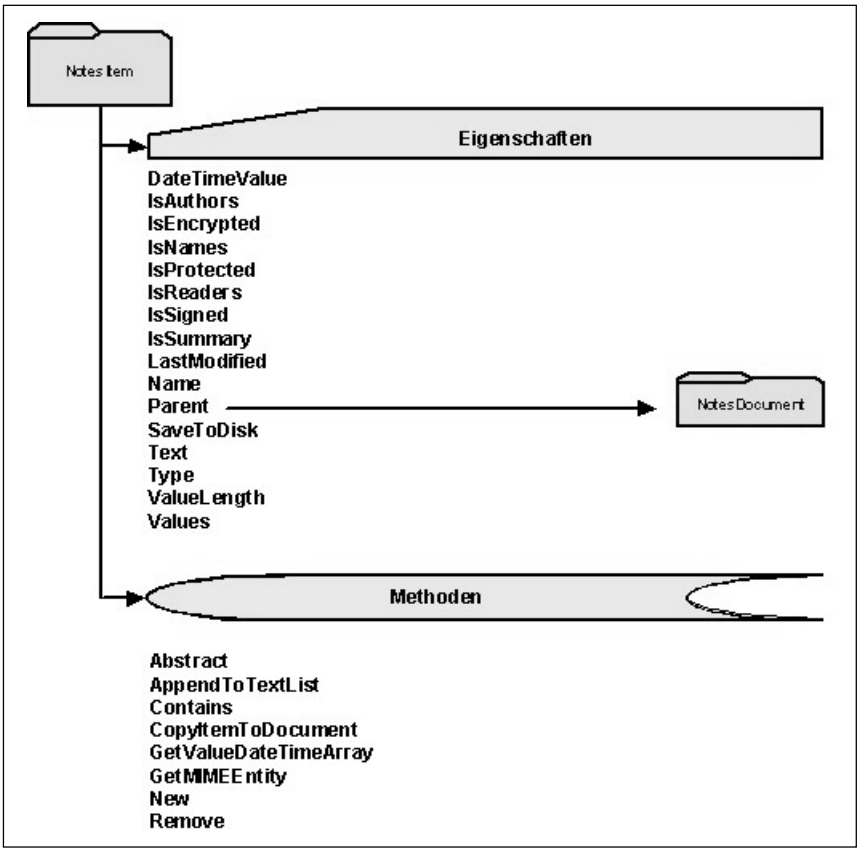


Auswahldialog für Textbausteine

Die wichtigste Erweiterung der NotesDocument-Klasse in Release 6 besteht in der Möglichkeit, Dokumente für einen Benutzer zu sperren. Damit reduziert sich die Anzahl der Fälle, dass ein Dokument von mehreren Benutzern gleichzeitig geändert werden kann. Für mobile Benutzer bleibt dieses Problem natürlich erhalten.

6.3.6.2.8 Die Klasse NotesItem

LotusScript unterscheidet zwischen „Items“, Elementen und Feldern innerhalb von Dokumenten. Items sind einzelne programmierbare Teile eines Dokuments.



Die Klasse NotesItem

Sie sind in der Regel mit Feldern identisch. Elemente werden in Release 6 neu eingeführt und dienen der Positionierung des Cursors in einem Richtext-Feld. Zur Erstellung eines Objektes der NotesItem-Klasse kann die Methode „AppendItemValue“ der Klasse NotesDocument oder die Methode „New“ der Klasse NotesItem verwendetet werden. Die Klasse NotesItem verfügt über folgende Eigenschaften:

- **DateTimeValue**
  Die Eigenschaft „DateTimeValue“ enthält bei einem Datums-/Zeitfeld einen Wert mit Zeitzone und mit hundertstel Sekunden. Bei anderen Feldtypen enthält sie den Wert „Nothing“. Die Eigenschaft kann gelesen und auch geschrieben werden. Das Ergebnis ist ein Objekt der Klasse NotesDateTime.
  Set NotesDateTime = NotesItem.DateTimeValue
  NotesItem.DateTimeValue = NotesDateTime

- **IsAuthors**
  Zeigt an, ob es sich um ein Autorenfeld handelt. Ist das der Fall, ist der Zugriff das Dokument über Autorenrechte geregelt (nur die in diesem Feld aufgeführten Autoren haben Zugriff).
  BooleanVar = NotesItem.IsAuthors
  NotesItem.IsAuthors = BooleanVar

- **IsEncrypted**
  liefert zurück, ob das Feld verschlüsselt ist oder nicht. Wird die Eigenschaft auf „True“ gesetzt, muss die Methode „Verschlüsseln“ auf das Dokument angewendet werden.
  BooleanVar = NotesItem.IsEncrypted
  NotesItem.IsEncrypted = BooleanVar

- **IsNames**
  Die Eigenschaft „IsNames“ ist wahr, wenn es sich bei dem Item um ein Namensfeld handelt. In diesem Feld werden Benutzernamen gespeichert. Das hat keine Auswirkung auf die Zugriffsrechte des Dokumentes.
  BooleanVar = NotesItem.Names
  NotesItem.IsNames = BooleanVar

- **IsProtected**
  gibt an, ob ein Benutzer mindestens Editorrechte benötigt, um das Feld zu bearbeiten.
  BooleanVar = NotesItem.IsAuthors
  NotesItem.IsAuthors = BooleanVar

- **IsReaders**
  zeigt an, ob es sich um ein Leserfeld handelt. Ist das der Fall, ist der Zugriff des Dokumentes über Leserrechte geregelt (nur die in diesem Feld aufgeführten Leser haben Zugriff).
  BooleanVar = NotesItem.IsReaders
  NotesItem.IsReaders = BooleanVar

- **IsSigned**
  liefert zurück, ob das Feld eine Signatur enthält.
  BooleanVar = NotesItem.IsSigned
  NotesItem.IsSigned = BooleanVar

- **IsSummary**
  Über die Eigenschaft „IsSummary“ wird gesteuert, ob ein Feld in einer Ansicht oder einem Ordner angezeigt werden kann. Wird das Feld über die „New“-Methode der NotesItem-Klasse erstellt, enthält die Eigenschaft den Standardwert „False“.

Bei einer Erstellung über die „AppendItemValue“-Methode der Klasse NotesDocument enthält sie den Wert „True“. Felder mit Richtext als Inhalt haben immer den Wert „False“.
BooleanVar = NotesItem.IsSummary
NotesItem.IsSummary = BooleanVar

- **LastModified**
  enthält das Datum der letzten Änderung.
  variantDatum = NotesItem.LastModified

- **Name**
  Name des Items.
  StringName = NotesItem.Name

- **Parent**
  enthält das Dokument, in dem sich das Item als Objekt befindet.
  Set NotesDocument = NotesItem.Parent

- **SaveToDisk**
  Diese Eigenschaft steuert, ob ein bestimmtes Item gespeichert wird oder nicht. Ist es bereits gespeichert und die Eigenschaft „SaveToDisk“ auf „False“ gesetzt, dann wird das gespeicherte Item bei dem nächsten Speichervorgang von der Festplatte gelöscht.
  BooleanVar = NotesItem.SaveToDisk
  NotesItem.SaveToDisk = BooleanVar

- **Text**
  Textinhalt des Items. Der Inhalt von Richtext-Feldern wird in einfachen Text umgewandelt. Handelt es sich um ein Feld mit Mehrfachwerten, so werden diese durch Semikolon getrennt, aber in der gleichen Zeichenkette wiedergegeben.
  stringText = NotesItem.Text

- **Type**
  liefert den Datentyp des Items als Zahlenwert zurück.
  integerTyp = NotesItem.Type

- **ValueLength**
  Bytelänge des Items nach dem Speichern.
  longAnzahl = NotesItem.ValueLength

- **Values**
  Werte, die im Item gespeichert sind oder die ihm zugewiesen werden sollen.
  stringValue oder arrayValues = NotesItem.Values
  NotesItem.Values = arrayValues

Auch einige Methoden des Items können eingesetzt werden. Viele Methoden zur Bearbeitung von Items finden sich auch in der Klasse NotesDocument.

- **Abstract**
  Der Itemwert wird verkürzt (abstrahiert). Selbstlaute werden entfernt und Abkürzungen aus einer Wörterbuchdatei (notesabbr.txt) eingesetzt. Der Rückgabewert wird auf die angegebene Länge gekürzt.
  stringText = NotesItem.Abstract(longMaxLänge, _
  booleanErsetzen, booleanWörterbuchdatei)

- **AppendToTextList**
  fügt ein neues Element zu einem Item mit Mehrfachwerten hinzu.
  Call NotesItem.AppendToTextList(neuerWert)

- **Contains**
  überprüft ein Item mit Mehrfachwerten darauf, ob eines der Elemente genau dem Vergleichswert entspricht.
  booleanVar = NotesItem.Contains(stringVergleich)

- **CopyItemToDocument**
  erstellt eine Kopie des NotesItem-Objektes in einem anderen Dokument. Das neue NotesItem-Objekt kann dabei einen neuen Namen erhalten.
  Set NotesItem = _
  NotesItem.CopyItemToDocument(NotesDocument, _
  stringName)

- **GetMIMEEntity**
  liefert ein Objekt der Klasse NotesMIMEEntity aus einem Item.
  Set NotesMIMEEntity = NotesItem.GetMIMEEntity

- **GetValueDateTimeArray (ab Release 6)**
  übernimmt die NotesDateTime-Objekte aus einem Item in eine Array.
  arrayNotesDateTime = NotesItem.GetValueDateTimeArray

- **New**
  Bei „New“ handelt es sich um eine Methode zur Erstellung eines neuen Objektes der Klasse NotesItem. Bei der Erstellung wird das Dokument, der Name des Items, ein Wert sowie optional ein Datentyp angegeben.
  Set NotesItem = New NotesItem(NotesDocument, _
  stringName, Wert, [integerTyp])

- **Remove**
  löscht ein Item aus dem Dokument. Nach dem Speichern des Dokumentes ist das Item dort entfernt.
  Call NotesItem.Remove

Programmtext

```
'/// Routine zur Übernahme von e-Mail-Dokumenten in die
'/// aktuelle Datenbank

Sub getMemos(strFolder As String)
'/// Der Name des Ordners, aus dem die Dokumente
'/// übernommen werden sollen, wird als Parameter
'/// übergeben

'/// NotesSession-Objekt erstellen
    Dim ses As New NotesSession

    '/// NotesUiWorkSpace-Objekt erstellen
Dim ws As New NotesUiWorkSpace

    '/// Objekt für die Mail-Datenbank definieren
Dim dbMail As NotesDatabase

    '/// NotesDocumentCollection-Objekt erstellen für die
'/// ausgewählten e-Mail-Dokumente
Dim colMail As NotesDocumentCollection

    Dim vwMail As NotesView
'/// Ansichtsobjekt für den Ordner

'/// Dokumentobjekt für die Mail-Datenbank definieren
    Dim docMail As NotesDocument
```

```
'/// Dokument-Objekt für die aktuelle Datenbank definieren
    Dim doc As NotesDocument

'/// Item-Objekt definieren
    Dim item As NotesItem

    Dim rtitem As NotesRichTextItem
    Dim intCount As Integer
Dim intR As Integer
    Dim strMailFile As String
    Dim strMsg As String
    Dim strText As String
    Dim strPrint as String

    strPrint = "..."

'/// Wenn ein beliebiger Ordner verwendet werden soll,
'/// wird das Wort "Ordner" an die Prozedur übergeben.
'/// In diesem Fall wird der Name des Ordners vom
'/// Benutzer erfragt. Der Vorgabewert (netBook) kann
'/// beliebig geändert werden.
    If strFolder ="Ordner" Then
      strFolder = Inputbox$("Ordnername: ", "Lotus")
    End If

    '/// Name der Mail-Datenbank des Benutzers aus der
    '/// Datei "Notes.ini" lesen
    strMailFile = _
Cstr(ses.GetEnvironmentString("MailFile", True))

'/// Datenbankobjekt erstellen
    Set dbMail = New _
    NotesDatabase(ses.CurrentDatabase.Server, strMailFile)
```

```
'/// Ansichtsobjekt für den Ordner erstellen
    Set vwMail = dbMail.GetView(strFolder)

'/// Fehlermeldung ausgeben, wenn der Ordner nicht
'/// gefunden wird.
    If vwMail Is Nothing Then
      strMsg = "Ungültiger Ordnername!"  & _
      Chr(10) & Chr(10) & _
      "Der Ordner mit dem Namen" & _
      Chr(10) & Chr(10) & _
      strFolder & Chr(10) & Chr(10) & _
      "fehlt in Ihrer Maildatenbank!"
      Msgbox  strMsg, MB_ICONSTOP, "Lotus"

      '/// Prozedur beenden
      Exit Sub
    End If

'/// Dokumente in einem Picklist-Dialog auswählen und in eine
'/// Dokumentensammlung übernehmen
Set colMail = ws.PickListCollection(PICKLIST_CUSTOM, _
True, ses.CurrentDatabase.Server, strMailFile, strFolder, _
"Lotus", "Bitte wählen Sie: ")

    '/// Wenn kein Dokument ausgewählt wurde,
    '/// Fehlermeldung ausgeben und die Prozedur
    '/// verlassen
    If colMail.count = 0 then
      MsgBox "Kein Dokument übernommen.", _
      36, "Notes"
      Exit Sub
    End if
```

```
    intCount = 0

    '/// Erstes Dokument der Dokumentensammlung
    '/// öffnen
    Set docMail = colMail.getFirstDocument

    '/// wenn kein Dokument zur Verfügung steht,
    '/// Fehlermeldung ausgeben und Prozedur
    '/// verlassen
    if docMail is Nothing then
MsgBox "Keine neuen E-Mails im Ordner " & _
strFolder & " gefunden!"
      Exit sub
    End if

    '/// Verarbeitungsschleife für die ausgewählten
    '/// Dokumente
    While Not docMail Is Nothing
      '/// Neues Dokument anlegen
    Set doc = _
    ses.CurrentDatabase.CreateDocument

      With doc
          '/// Maske zuweisen
          .Form = "frmBrief"

          '/// Item-Objekt für das Feld Subjekt
          '/// in der Maildatenbank erstellen
          Set item = docMail.GetFirstItem("Subject")

          '/// Wenn das Item gefunden wurde,
          '/// wird es in das neue Dokument
          '/// kopiert
```

```
If Not item Is Nothing Then
        item.CopyItemToDocument _
        doc, “Subject”
End If

‘/// Weiteren Feldern des neuen
‘/// Dokumentes Werte zuweisen
.UserFormName = “Memo”
.FormularTyp = “Memo”
.anoDocAutor = ses.UserName
.MailForm = True
.FormularName = “Memo”
.FormFirmaGroup = _
.strStatus = “Offen”
.DokArt = “E-Mail”
.Firmenzeile = “>>> Neues E-Mail <<<”
Set item = _
docMail.GetFirstItem(“Principal”)

If Not item Is Nothing Then
        item.CopyItemToDocument _
        doc, “NotAutor”
        item.CopyItemToDocument _
        doc, “From”
        strText = Lcase(item.Values(0))
        .Personenzeile = strText
        .MailAdress = strText
        .SendTo = strText
Else
        Set item = _
        docMail.GetFirstItem(“From”)
        item.CopyItemToDocument _
```

```
                doc, "NotAutor"
                item.CopyItemToDocument _
                doc, "From"
                strText = Lcase(item.Values(0))
                .Personenzeile = strText
                .EMailSB = strText
                .SendTo = strText
            End If

            .datRemind = Today
            .datEnd = Today
            .datWriteDay = docMail.PostedDate
            .Ersteller = "ITP"
            .Bearbeiter = ses.CommonUserName
            .Principal = ses.UserName
            .FolgeAktion = "Bearbeiten"

            '/// Body-Feld mit dem eigentlichen
            '/// e-Mail-Text übernehmen
            Set rtitem = docMail.GetFirstItem("Body")

            If Not item Is Nothing Then
                rtitem.CopyItemToDocument _
                doc, "Body"
            End If

            .save True, False
        End With

        Set docMail = _
        colMail.GetNextDocument(docMail)
        strPrint = strPrint & "."
        Print intCount; " E-Mails übernommen " & strPrint
    Wend
```

```
    If intCount = 0 Then
      strMsg = "Es wurde kein Dokument übernommen."
      intR = 0
    Elseif intCount = 1 Then
      strMsg = _
      "Es wurde ein Dokument übernommen." & _
      Chr$(13) & Chr$(10) & Chr$(13) & Chr$(10) & _
      " Dokument der Mail-Datenbank löschen?"
      intR = Msgbox(strMsg, 36, "Lotus")
    Else
      strMsg = "Es wurden " & Str$(intCount) & _
      " Dokumente übernommen." & _
      Chr$(13) & Chr$(10) & Chr$(13) & Chr$(10) & _
      "Dokumente in der löschen?"
      intR = Msgbox(strMsg, 36, "Lotus")
    End If

    If intR = 6 Then
      Print "Lösche ..."
      colMail.RemoveAll True
    End if

    Print
    ws.ViewRefresh

'///
End Sub
```

6.3.6.2.9 Die Klasse NotesDocumentCollection

Eine der häufigsten Ausgangssituationen bei einer Domino-Anwendung besteht darin, eine Anzahl von Dokumenten nach bestimmten Kriterien auszuwählen und diese Dokumente dann in einer speziellen Form zu bearbeiten. Einfach ist es, wenn bereits eine Ansicht oder ein Ordner zur Verfügung stehen, die der gewünschten Dokumentenauswahl entsprechen. Häufig ist das aber nicht der Fall. Es muss eine Anweisungsfolge erstellt werden, die alle gesuchten Dokumente auswählt und sie dann zur Weiterverarbeitung bereitstellt. In LotusScript stellen Objekte der Klasse NotesDocumentCollection einen Auffang-Container dar, der diese Aufgabe übernimmt und auch Methoden und Eigenschaften kennt, mit denen die Dokumente bearbeitet werden können.

NotesDocumentCollection	
Eigenschaften	***Ergebnis-Objekte***
Count	
IsSorted	
Parent	NotesDatabase
Query	
Methoden	
AddDocument	
DeleteDocument	
FTSearch	NotesDocumentCollection
GetDocument	NotesDocument
GetFirstDocument	NotesDocument
GetLastDocument	NotesDocument
GetNextDocument	NotesDocument
GetNthDocument	NotesDocument
GetPrevDocument	NotesDocument
PutAllInFolder	
RemoveAll	
RemoveAllFromFolder	
StampAll	
UpdateAll	

Während die Dokumente einer Ansicht in der Regel eine bestimmte Sortierung aufweisen, sind die Dokumente eines NotesDocumentCollection nur dann sortiert, wenn sie mit Hilfe eines Volltextindexes erstellt wurden.

Eine NotesDocumentCollection wird aus den Dokumenten einer Datenbank zusammengestellt. Deshalb sind es im Wesentlichen Methoden und Eigenschaften der NotesDatabase-Klasse, die dazu verwendet werden, ein solches Objekt zu erstellen:

- **AllDocuments**
  Diese Eigenschaft enthält eine Liste mit allen Dokumentobjekten der Datenbank.

- **UnprocessedDocuments**
  Diese Eigenschaft enthält eine Liste aller Dokumentobjekte, die ausgewählt und noch nicht verarbeitet wurden.

- **(FT-)Search**
  Methode, die eine Dokumentensammlung über einen Such-String zusammenstellt (FTSearch bedient sich dabei eines Volltextindexes).

- **Unprocessed(FT)Search**
  Nur ausgewählte Dokumente, die einem Such-String entsprechen, werden in eine Dokumentensammlung übernommen.

Auch andere Notes-Klassen bieten Möglichkeiten, solche Dokumentsammlungen zu erstellen (z. B. die NotesView-Klasse mit der Methode „GetAllDocumentsByKey“).

Ist eine Dokumentensammlung erstellt, verfügt sie über folgende Eigenschaften:

- **Count**
  Anzahl der Dokumentobjekte, die in der Dokumentensammlung enthalten sind. Enthält die Sammlung keine Dokumente, enthält die Count-Eigenschaft den Wert 0.
  longAnzahl = NotesDocumentCollection.Count

- **IsSorted**
  Diese Eigenschaft enthält normalerweise den Wert „False“. Sie ist also unsortiert. Nur wenn die Dokumentensammlung mit Hilfe eines Volltextindexes erstellt wurde, ist sie sortiert und die IsSorted-Eigenschaft enthält den Wert „True“.
  booleanWert = NotesDocumentCollection.IsSorted

- **Parent**
  Das Datenbankobjekt der Dokumente, die in der Sammlung aufgeführt sind, kann über diese Eigenschaft abgerufen werden.
  Set NotesDatabase = NotesDocumentCollection.Parent

- **Query**
  Der Inhalt der Eigenschaft „Query“ besteht aus einer Zeichenkette, die dem Such-String entspricht, mit dessen Hilfe die Dokumentensammlung erstellt wurde. Wenn die Sammlung nicht über eine Suche, sondern über eine Eigenschaft des Datenbankobjektes zusammengestellt wurde, enthält „Query“ einen leeren String.
  stringSuche = NotesDocumentCollection.Query

Die Methoden der NotesDocumentCollection-Klasse dienen dazu, auf die Dokumente der Klasse zuzugreifen und sie gegebenenfalls zu bearbeiten:

- **AddDocument**
  wird dazu verwendet, einer Dokumentensammlung ein neues Dokument hinzuzufügen.
  Call _NotesDocumentCollection.AddDocument(NotesDocument)

- **DeleteDocument**
  entfernt ein bestimmtes Dokument aus der Dokumentensammlung. Das Dokument selbst wird dadurch nicht gelöscht.
  Call _NotesDocumentCollection.DeleteDocument(NotesDocument)

- **FTSearch**
  Diese Methode erlaubt die Suche nach Dokumenten, die einem bestimmten Such-String entsprechen – innerhalb einer Dokumentensammlung. Dabei sollte für die zugrundeliegende Datenbank ein Volltextindex vorhanden sein. Nach der Suche enthält die Sammlung nur noch Dokumente, die dem Such-String entsprechen. Mittels des Parameters „integerAnzahl" kann die maximale Anzahl an Dokumenten festgelegt werden, die in der neuen Sammlung enthalten sein sollen. Wird als „integerAnzahl" der Wert 0 angegeben, werden alle auf den Such-String zutreffenden Dokumente zurückgeliefert.
  Call NotesDocumentCollection.FTSearch(stringSuch, _ integerAnzahl)

- **GetDocument**
  erstellt ein NotesDocument-Objekt aus dem angegebenen Dokument. Als Ergebnis zeigen zwei Objekte auf das gleiche Dokument.
  Set NotesDocument = _
  NotesDocumentCollection.GetDocument(NotesDocument)

- **GetFirstDocument**
  Die Methode „GetFirstDocument“ stellt die häufigste Form des Zugriffs auf eine Dokumentensammlung dar. In der Regel wird eine Dokumentensammlung erstellt und dann mit Hilfe einer Schleife verarbeitet. Mit „GetFirstDocument“ wird der Zeiger auf das erste Dokument gesetzt. Von dort aus werden alle Dokumente nacheinander mit den Anweisungen einer Schleife bearbeitet.
  NotesDocument = _
  NotesDocumentCollection.GetFirstDocument

- **GetLastDocument**
  setzt den Verarbeitungszeiger auf das letzte Dokument der Sammlung.
  NotesDocument = _
  NotesDocumentCollection.GetLastDocument

- **GetNextDocument**
  erstellt ein NotesDocument-Objekt aus dem nächsten Dokument der Sammlung. Damit das Dokument gefunden werden kann, muss das aktuelle Dokument angegeben werden.
  NotesDocument = _
  NotesDocumentCollection.GetNextDocument(NotesDocument)

- **GetNthDocument**
  liefert ein NotesDocument-Objekt, das dem n-ten Dokument der Sammlung entspricht. Diese Methode wird häufig dann verwendet, wenn die Verarbeitung über eine For-Next-Schleife erfolgt.
  Set NotesDocument = _
  NotesDocumentCollection.GetNthDocument(integerNummer)

- **GetPrevDocument**
  stellt ein NotesDocument-Objekt des vorherigen Dokumentes bereit. Dazu muss auch wieder das aktuelle Dokument als Parameter übergeben werden.
  NotesDocument = _
  NotesDocumentCollection.GetPrevDocument(NotesDocument)

- **PutAllInFolder**
  nimmt alle Dokumente der Sammlung in den angegebenen Ordner auf. Existiert der angegebene Ordner nicht in der verwendeten Datenbank, so wird er automatisch angelegt, wenn der Parameter „booleanAnlegen“ mit “True” belegt wird.
  Call NotesDocumentCollection.PutAllInFolder(stringOrdner, _ booleanAnlegen)

- **RemoveAll**
  Diese Methode ist mit höchster Vorsicht zu behandeln. Sie löscht alle Dokumente der Sammlung. Damit sind diese Dokumente auf dem Datenträger gelöscht. Der Parameter „booleanSofort“ sorgt für eine unverzügliche Durchführung.
  Call NotesDocumentCollection.RemoveAll(booleanSofort)

- **RemoveAllFromFolder**
  entfernt die Dokumente der Sammlung aus dem genannten Ordner.
  Call NotesDocumentCollection.RemoveAllFromFolder(_ stringOrdner)

- **StampAll**
  Mit „StampAll“ können Werte innerhalb von Dokumenten geändert werden, die in einer Dokumentensammlung enthalten sind. Es werden die echten Dokumente und nicht die NotesDocument-Objekte der Sammlung geändert. Das hat zur Folge, dass keine „Save“-Methode benötigt wird, um die Werte in die Dokumente zu übertragen. Das führt sogar dazu, dass ein “Save” den geänderten Wert zurücksetzt. Die Verwendung von “StampAll” sollte also nur dann erfolgen, wenn die echten Dokumente auf dem neuesten Stand sind. Soll mit den NotesDocument-Objekten der Dokumentensammlung weitergearbeitet werden, so muss ihr Inhalt aus den Dokumenten aktualisiert werden. Der Daten-Typ des neuen Wertes muss dem des „NotesItems“ entsprechen.
  Call NotesDocumentCollection.StampAll(stringNotesItem, _ neuerWert)

- **UpdateAll**
  markiert alle Dokumente der Sammlung als verarbeitet. Damit werden sie in einem weiteren Auswahlaufruf nicht mehr als unverarbeitet erkannt.
  Call NotesDocumentCollection.UpdateAll

Programmtext

```
‘/// Beispiel 1
‘/// Der Status aller ausgewählten Dokumente wird durch
‘/// einen vom Benutzer angegebenen Status ersetzt.

Sub changeStatus(strNewStatus As String)
‘///

‘/// NotesUiWorkSpace-Objekt erstellen
    Dim ws As New NotesUiWorkspace

    ‘/// NotesSession-Objekt erstellen
    Dim ses As New NotesSession

    ‘/// NotesDocumentCollection-Objekt definieren
    Dim col As NotesDocumentCollection

    ‘/// NotesDocument-Objekt definieren
    Dim doc As NotesDocument

    ‘/// Zählvariable definieren
    Dim i As Integer

    ‘/// Aus der aktuellen Datenbank werden alle
‘/// ausgewählten Dokumente in die Sammlung
‘/// übernommen
    Set col = ses.CurrentDatabase.UnprocessedDocuments

    ‘/// Neuen Status abfragen
strNewStatus = Inputbox$(“Neuer Status:”, “netOffice”, _
strNewStatus, 1000, 1000)
```

```
        '/// Schleife mit so vielen Durchläufen, wie
    '/// Dokumente in der Sammlung enthalten
    '/// sind
        For i = 1 To col.Count

            '/// Das nächste Dokument auswählen
            Set doc = col.GetNthDocument( i )

            '/// Statusfeld ändern
            doc.strStatus = strNewStatus

            '/// Dokument ändern
            doc.Save True, False
        Next i

        '/// Status-Ansicht aktualisieren
        ws.ViewRefresh
    '///
    End Sub


'/// Beispiel 2
'/// Über einen Suchstring werden Dokumente ausgewählt
'/// und ein Feld geändert

Sub changeForm
'///
    Dim ws As New NotesUiWorkspace
    Dim doc As NotesDocument
    Dim col As NotesDocumentCollection
```

```
    '/// Startdatum für die Search-Methode
    '/// definieren
    Dim datStart As New NotesDateTime("1.1.1990")

    '/// Suchstring definieren
Dim strSuch As String

    '/// Inhalt des Suchstrings festlegen
    strSuch = |form = "frmBrief":"frmPage"|

    '/// Dokumente für die Sammlung ermitteln
Set col = ws.CurrentDatabase.Database.search(strSuch, _
datStart, 0)

    '/// Nur wenn Dokumente gefunden wurden, die dem
    '/// Suchstring entsprechen, wird die Verarbeitung
    '/// gestartet
    If col.count > 0 Then

      '/// Erstes Dokument ermitteln und
      '/// NotesDocument-Objekt erstellen
      Set doc = col.getFirstDocument

      '/// Die Schleifenverarbeitung wird
'/// beendet, sobald kein Dokument mehr
      '/// gefunden wird.
      While Not doc Is Nothing

          '/// Feld ändern
          doc.form = "frmDokument"

          '/// Dokument ändern
          doc.save True, False
```

```
            '/// Nächstes Dokument
            Set doc = col.getNextDocument(doc)
        Wend

    End If
'///
End Sub
```

6.3.6.2.10 Die Klasse NotesRichTextItem

Neben den einfachen Textfeldern kennt Notes einen Feldtyp, der es erlaubt, als Container für unterschiedliche Texte und Objekte zu dienen: „Richtext“. Während Felder normalerweise nur einheitliche Daten enthalten, können in Richtextfeldern die Texte mit unterschiedlichen Eigenschaften versehen und zusammen mit Bildern, Dateianhängen oder z.B. Soundobjekten gespeichert werden. In LotusScript wird zur Arbeit mit Richtextfeldern die Klasse NotesRichTextItem verwendet. Zur Arbeit mit den verschiedenen Texteigenschaften und Objekten innerhalb eines Richtextfeldes stehen weitere Klassen zur Verfügung.

Zur Erstellung eines Objektes der Klasse NotesRichTextItem kann die „New“-Methode oder die Methode „CreateRichTextItem“ der Klasse NotesDocument verwendet werden.

Die Klasse stellt folgende Eigenschaften und Methoden bereit:

NotesRichTextItem	
Eigenschaften	**Objekte**
DateTimeValue	NotesDateTime
EmbeddedObjects	Array mit NotesEmbeddedObject
IsAuthors	
IsEncrypted	
IsNames	
IsProtected	
IsReaders	
IsSigned	
IsSummary	
LastModified	
Name	
Parent	NotesDocument
SaveToDisk	
Text	
Type	
ValueLength	
Values	
Methoden	
Abstract	
AddNewLine	
AddPageBreak	
AddTab	

AppendDocLink	
AppendParagraphStyle	
AppendRTItem	
AppendStyle	
AppendTable	
AppendText	
AppendToTextList	
BeginInsert	
BeginSection	
Compact	
Contains	
CopyItemToDocument	NotesItem
CreateNavigator	NotesRichTextNavigator
CreateRange	NotesRichTextRange
EmbedObject	NotesEmbeddedObject
EndInsert	
EndSection	
GetEmbeddedObject	NotesEmbeddedObject
GetFormattedText	
GetMIMEEntity	NotesMIMEEntity
GetNotesFont	
GetUnformattedText	
GetValueDateTimeArray	NotesDateTimeArray
New	NotesRichtextItem
Remove	
Update	

Die Klasse NotesRichTextItem ist von der Klasse NotesItem abgeleitet. Sie erbt damit alle Eigenschaften und Methoden dieser Klasse. Folgende Eigenschaften sind mit der Klasse NotesRichTextItem verbunden:

- **DateTimeValue (siehe NotesItem)**
- **EmbeddedObjects**
  Die Eigenschaft „EmbeddedObjects“ enthält ein Array mit allen Objekten, die in dem Richtext-Objekt enthalten sind. Es kann sich dabei um OLE-Objekte, Verknüpfungen oder Datei-Anhänge handeln.
  arrayNotesEmbeddedObjects = _
  NotesRichtextItem.EmbeddedObjects

Programmtext

```
'/// Routine zum Ablösen von Dateien, die als Dateianhang in
'/// einem Richtextfeld gespeichert sind. Es wird davon
'/// ausgegangen, dass Dokumente ausgewählt wurden,
'/// deren Dateianhänge gespeichert werden sollen.

Sub saveAttachments
'///
  Dim ses As New NotesSession
  Dim ws As New NotesUiWorkspace
  Dim db as NotesDatabase

'/// Dokumentensammlung definieren für die ausgewählten
'/// Dokumente
  Dim col As NotesDocumentCollection
  Dim doc As NotesDocument
```

```
'/// Variantvariable für das Body-Feld definieren
Dim rtItem As Variant
'/// Objekt für die eingebetteten Dateien definieren
Dim objAtt As NotesEmbeddedObject
Dim i As Integer
Dim strFileName As String

'/// Pfadvariable für den Speicherort der Dateien erstellen
Dim strPfad As String
Dim varPfad As Variant

Set db = ses.CurrentDatabase

'/// Dokumentensammlung für die ausgewählten
'/// Dokumente erstellen
Set col =db.UnprocessedDocuments

'/// Fehlermeldung und Abbruch, wenn kein Dokument
'/// ausgewählt wurde
If col.Count = 0 Then
   Msgbox "Bitte Dokument auswählen!", 16, "Achtung!"
   Exit Sub
End If

'/// Verzeichnis als Speicherort erfragen
strPfad = Trim(Inputbox$ ( "Bitte das Verzeichnis angeben:", _
  "Achtung!", strPfad ))

'/// Fehlermeldung und Abbruch, wenn kein Verzeichnis
'/// angegeben wurde
If strPfad = "" Then
   Msgbox "Kein Verzeichnis!", 16, "Achtung!"
Exit Sub
```

```
'/// Backslash ergänzen, falls nicht angegeben
If Right(strPfad, 1) <> "\" Then strPfad = strPfad  & "\"

'/// Alle Dokumente nacheinander verarbeiten
For i = 1 To col.Count
  '/// Dokumentvariable erstellen
   Set doc = col.GetNthDocument(i)

  '/// Prüfen, ob ein eingebettetes Objekt in dem
  '/// Dokument enthalten ist
   If doc.HasEmbedded Then _
     '/// Objekt mit dem Body-Feld erstellen
      Set rtItem = doc.GetFirstItem("Body")

     '/// Prüfen, ob es sich um ein Richtextfeld handelt
      If ( rtItem.Type = RICHTEXT ) Then

        '/// Jedes eingebettete Objekt verarbeiten
         Forall o In rtItem.EmbeddedObjects

      '/// Wenn es sich um einen Dateianhang handelt
       If ( o.Type = EMBED_ATTACHMENT ) Then
         '/// Anhang speichern
          o.ExtractFile strPfad & o.Source
      End If

        End Forall

     End If

  End If

Next If
```

```
    '/// Abschlussmeldung ausgeben
     Print Str$(col.Count) & " Dateien nach " & strPfad & _
       " gespeichert ..."

  '///
   End Sub
```

- **IsAuthors (s. NotesItem)**
- **IsEncrypted (s. NotesItem)**
- **IsNames (s. NotesItem)**
- **IsProtected (s. NotesItem)**
- **IsReaders (s. NotesItem)**
- **IsSigned (s. NotesItem)**
- **IsSummary (s. NotesItem)**
- **LastModified (s. NotesItem)**
- **Name (s. NotesItem)**
- **Parent (s. NotesItem)**
- **SaveToDisk (s. NotesItem)**
- **Text (s. NotesItem)**
- **Type (s. NotesItem)**
- **ValueLength (s. NotesItem)**
- **Values (s. NotesItem)**

Auch die Methoden der Klasse NotesItem sind für die Objekte der Klasse NotesRichTextItem verfügbar. Dazu kommen Methoden, die für die speziellen Erfodernisse dieser Klasse ergänzt wurden:

- **Abstract (s. NotesItem)**

- **AddNewLine**
  Mithilfe der Methode „AppendNewLine“ können eine oder mehrere Zeilenschaltungen an den Inhalt eines Richtext-Feldes angehängt werden. Mit „BeginInsert“ und „EndInsert“ kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AddNewLine(integerZahl, booleanAbsatz)
  „IntegerZahl“ gibt die Anzahl der Zeilenschaltungen an, die eingefügt werden sollen. „booleanAbsatz“ steuert, ob nach der Zeilenschaltung ein neuer Absatz beginnen soll.

- **AddPageBreak**
  fügt einen Seitenwechsel am Ende des Feldinhaltes an. Mit „BeginInsert“ und „EndInsert“ kann die Einfügeposition beeinflusst werden.
  Call NotesRichtextItem.AddPageBreak _
  ([NotesRichTextParagraphStyle])
  Das angegebene „NotesParagraphStyle“-Objekt steuert die Eigenschaften des Absatzes, mit dem die neue Seite beginnt.

- **AddTab**
  hängt ein oder mehrere Tabulatorzeichen an den Text an. Mit „BeginInsert“ und „EndInsert“ kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AddTab(integerAnzahl)

- **AppendDocLink**
  Wenn eine Verknüpfung zu einem Dokument, einer Ansicht oder einer Datenbank in ein Richtextfeld eingefügt werden soll, kann das mithilfe der „AppendDocLink"-Methode geschehen. Als Parameter werden angegeben: das Objekt für die Verknüpfung, ein Kommentar und ein Text, der als „Hotspot" angezeigt wird.
  Call NotesRichTextItem.AppendDocLink(objxxx, _
  stringKommentar [, stringHotspot])
  Dabei bedeutet xxx ein Dokument-, Ansichts- oder Datenbankobjekt. Mit „BeginInsert" und „EndInsert" kann die Einfügeposition beeinflusst werden.

- **AppendParagraphStyle**
  Soll der Stil eines Textabsatzes neu festgelegt werden, bevor er eingefügt wird, so kann das mithilfe der „AppendParagraphStyle"-Methode erfolgen. Mit „BeginInsert" und „EndInsert" kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AppendParagraphStyle(_
  NotesRichTextParagraphStyle)

- **AppendRTItem**
  fügt ein Richtextfeld in ein Richtextfeld ein. Mit „BeginInsert" und „EndInsert" kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AppendRTItem(NotesRichTextItem)

- **AppendStyle**
  ändert die Textattribute des folgenden Textes. Mit „BeginInsert" und „EndInsert" kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AppendStyle(NotesRichTextStyle)

Programmtext

```
'/// Dokument anlegen und Felder intialisieren. Das Body-Feld
'/// soll bereits mit einem Starttext gefüllt sein.
'/// Die Prozedur wird z.B. über den "Click"-Event einer
'/// Schaltfläche gestartet. Über ein Steuerungsformular können
'/// Vorgabewerte von Benutzern gefüllt und in die Dokumente
'/// übernommen werden.

Sub initDocument(strForm As String)
'///
  Dim ses As New NotesSession
  Dim ws As New NotesUiWorkspace
  Dim db As NotesDatabase
  Dim doc As NotesDocument

 '/// Objekt für das Richtextfeld definieren
  Dim rtItem As NotesRichTextItem

 '/// Objekt für die Textattribute definieren
  Dim rtStyle As NotesRichTextStyle

 '/// Objekt für die Absatzeigenschaften definieren
  Dim rptStyle As NotesRichTextParagraphStyle

 '/// Variable für Beispieltexte
  Dim strKurz As String
  strAbsFunktion As String
  strAbsKurz As String
  strAbsUnterschrift As String
  Dim strPS As String
```

```
'/// Objekt für die aktuelle Datenbank erstellen
Set db = ses.CurrentDatabase

'/// Dokument-Objekt anlegen
Set doc = New NotesDocument(db)

'/// Name der Maske festlegen
doc.Form = "frmBrief"

'/// An dieser Stelle könnten Daten aus Vorgabedokumenten
'/// übernommen werden.
'/// Das könnten Standardwerte eines Formlares oder auch
'/// Daten des Benutzers sein.
'/// In diesem Beispiel wurden diese Werte aus Gründen der
'/// Übersichtlichkeit fest eingesetzt.

strKurz = "pp/tg"
strAbsFunktion = "Finanzbuchhaltung"
strAbsKurz = "i.A."
strAbsUnterschrift = "Peter Muster"
strPS = "PS: Antworten Sie bitte unverzüglich!"

'/// Objekt für Textattribute erstellen
Set rtStyle = ses.createRichTextStyle

'/// Objekt für Absatzeigenschaften erstellen
Set rtpStyle = session.CreateRichTextParagraphStyle
'/// Objekt für das Body-Feld erstellen
Set rtItem = doc.CreateRichTextItem("Body")

'/// Absatzeigenschaften festlegen
rtpStyle.InterLineSpacing = SPACING_DOUBLE
richText.AppendParagraphStyle rtpStyle
```

```
'/// Textattribute festlegen
rtStyle.Bold = False
rtStyle.NotesFont = FONT_ROMAN
rtStyle.FontSize = 12
rtItem.AppendStyle rtStyle

'/// Inhalt des Richtextfeldes zusammenstellen
rtItem.AppendText "Hallo "
rtItme.AddNewLine(4)
rtItem.AppendText Trim(strAbsKurz & " " & strAbsUnterschrift)
rtItem.AddNewLine(2)
rtItemForm.AppendText strPS

'/// Weitere Felder initialisieren
doc.datWriteDay = Today
doc.Bearbeiter = ses.CommonUserName

'/// Bestehende Rahmen-Einstellungen aufheben
ws.setTargetFrame ""

'/// Neues Dokument im Bearbeitungsmodus öffnen
ws.editDocument True, doc

'///
End Sub
```

- **AppendTable (Release 6)**
  Viele Aufgaben erfordern es, Text in bestimmten Bereichen einer Seite zu positionieren. Ein Instrument zur Erreichung dieses Zieles stellt die Tabelle dar. Der Text wird über die Anordnung der Zellen positioniert und zusätzlich können Summenwerte oder andere Ergebnisse gezielt an der passenden Stelle angezeigt werden. Mit der Methode „AppendTable" können jetzt Tabellen in ein Richtext-Feld integriert werden. Mit „BeginInsert" und „EndInsert" kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AppendTable(integerZeilen, _ integerSpalten [, arrayBezeichnung] [, longLinkerRand] _ [,arrayAbsatzStil])
  In einem Array mit Zeichenketten werden die Bezeichnungen für den Fall hinterlegt, dass eine Tabelle mit Registerreitern erstellt wird. Zusätzlich kann für jede Spalte ein Absatzstil in Form eines Arrays definiert werden.

- **AppendText**
  Mithilfe dieser Methode wird Text in das Richtext-Feld eingefügt. Die Texteigenschaften des umgebenden Textes werden dabei übernommen. Mit „BeginInsert" und „EndInsert" kann die Einfügeposition beeinflusst werden.
  Call NotesRichTextItem.AppendText(stringText)

- **AppendToTextList (s. NotesItem)**

- **BeginInsert (Release 6)**
  Die „BeginInsert"-Methode stellt die Einfügeposition vom Ende des bestehenden Inhaltes auf eine neue Position um. Als Position können bestehende Objekte im Richtext-Feld verwendet werden.

Call NotesRichTextItem.BeginInsert(objxxx, booleanDanach)
Als Objekt kommen in Frage:
– NotesEmmbeddedObject
– NotesRichTextDocLink
– NotesRichTextNavigator
– NotesRichTextRange
– NotesRichTextSection
– NotesRichTextTable
Der zweite Parameter legt fest, ob die Einfügeposition vor oder hinter dem angegebenen Objekt liegen soll.

- **BeginnSection (Release 6)**
  Notes kennt eine spezielle Form von Textbereichen: die Abschnitte. Diese Bereiche sind daran zu erkennen, dass sie farblich gekennzeichnet und ein- bzw. ausgeklappt werden können. Im eingeklappten Zustand ist von dem Absatz nur die Überschrift oder sogar nur ein Hinweisdreieck zu sehen. Seit Release 6 können solche Abschnitte als Objekt in ein Richtext-Feld eingefügt werden.
  Call notesRichTextItem.BeginSection(stringTitel, _
  [NotesRichTextStyle], [NotesColorObject], _
  [booleanAufklappen])
  Als Parameter können angegeben werden:
  – eine Überschrift für den Abschnitt, der angezeigt wird, wenn der Abschnitt eingeklappt ist,
  – ein Absatzstil für die Überschrift,
  – eine Farbe für die Kennzeichnung der Überschrift,
  – ein Wahrheitswert dafür, ob der Abschnitt auf- oder eingeklappt angezeigt werden soll.
  Mit „BeginInsert“ und „EndInsert“ kann die Einfügeposition beeinflusst werden.

- **Compact (Release 6)**
  Da ein Richtext-Feld die unterschiedlichsten Objekte enthalten kann, ist es möglich, dass durch häufiges Ändern ungenutzter Speicherplatz entsteht. Durch Einsatz der „Compact"-Methode kann dieser ungenutzte Platz entfernt werden. Es empfiehlt sich, diesen Vorgang ans Ende der Verarbeitung – also z.B. vor die „Save"-Methode – zu setzen.
  Call NotesRichTextItem.Compact

- **Contains (s. NotesItem)**

- **CopyToDocument (s. NotesItem)**

- **CreateNavigator (Release 6)**
  erstellt ein Navigatorobjekt zu einem RichtextItem. Das Navigatorobjekt kann von verschiedenen anderen Methoden zur Steuerung der Einfügeposition verwendet werden.
  Set NotesRichTextNavigator = _
  NotesRichTextItem.CreateNavigator

- **CreateRange (Release 6)**
  Mithilfe der Methode "CreateRange" wird ein Bereichsobjekt erstellt, das zum Einfügen von Elementen und Attributen verwendet werden kann. Der Text des Bereiches kann ermittelt und weiterverarbeitet werden.
  Set NotesRichTextRange = NotesRichTextItem.CreateRange

- **EmbedObject**
  Fügt ein OLE-Objekt, eine Verknüpfung oder einen Datei-Anhang in ein Richtext-Feld ein.

Set NotesEmbeddedObject = _
notesRichTextItem.EmbedObject(integerArt, stringKlasse, _
stringQuellPfad[, stringName])
Bei den Typen wird unterschieden zwischen:
– EMBED_ATTACHMENT (1454) – Dateianhang
– EMBED_OBJECT (1453) – OLE-Objekt
– EMBED_OBJECTLINK (1452) – Verknüpfung
Die Zeichenkette mit dem Namen einer Klasse wird für OLE-Objekte verwendet, um die Verbindung zu dem entsprechenden Programm zu schaffen. Der Quellpfad wird bei Verknüpfungen und Dateianhängen ausgefüllt. Der Name kann eingesetzt werden, um das Objekt später wieder zu referenzieren.

Programmtext

```
'/// Dateien in ein Notes-Dokument als Dateianhang einbetten.
'/// Die Prozedur kann z.B. über eine Schaltfläche einer Ansicht
'/// gestartet werden. Soll ein bestehendes Dokument verwendet
'/// werden, muss zunächst über eine Dokumentensammlung
'/// das ausgewählte Dokument ermittelt werden. Im Beispiel
'/// wird unterstellt, dass ein neues Dokument angelegt wird.

Sub attachFiles
'///
  Dim ws As New NotesUiWorkspace
  Dim ses As New NotesSession
  Dim db As NotesDatabase
  Dim doc As NotesDocument
  Dim strPfad As String
  Dim rtItem As Variant
  Dim strFile As String
```

```
Set db = ses.CurrentDatabase

‘/// Objekt für Dokument erstellen
Set doc = db.CreateDocument

‘/// Verzeichnis aus der Datei „Notes.ini“ ermitteln.
‘/// Es könnte auch aus einer anderen Parameterdatei
‘/// oder über eine Abfrage ein Startverzeichnis
‘/// bestimmt werden.
strPfad = ses.getEnvironmentString(“Directory”, True)

‘/// Über den “Datei öffnen”-Dialog werden die Dateien
‘/// ermittelt, die als Anhang gespeichert werden sollen.
varFiles = ws.OpenFileDialog( True, _
  “Wählen Sie Dateien aus: “, “Alle Dateien | *.*”, strPfad)

‘/// Wenn keine Datei ausgewählt wurde, wird die Prozedur
‘/// beendet.
If Isempty(varFiles) Then Exit Sub

‘/// In dem neuen Dokument werden jetzt Standardfelder
‘/// belegt.
With doc
  .Form = “frmDokument”
  .Subject = “Datei-Anhänge”
  .datWriteDay = Today

  ‘/// Objekt für das Body-Feld erstellen
  Set rtitem = .createRichTextItem(“Body”)

  ‘/// Dateien anhängen
  Forall f In varFiles
    rtitem.embedObject EMBED_ATTACHMENT, “”, f
  End Forall
```

```
    '/// Dokument speichern
    .save True, False
  End With

  '/// Ansicht aktualisieren
  ws.ViewRefresh
'///
End Sub
```

- **EndInsert (Release 6)**
  Dient zur Kennzeichnung eines Bereiches, der mit „BeginInsert" eingeleitet wurde.
  Call NotesRichTextItem.EndInsert

- **EndSection (Release 6)**
  Schließt einen Abschnitt ab, der mit „BeginSection" eingeleitet wurde.
  Call NotesRichTextItem.EndInsert

- **GetEmbeddedObject**
  Diese Methode liefert ein Objekt, das in einem Richtext-Feld enthalten ist. Zur Auffindung des Objektes wird der Name verwendet, der bei der Methode „EmbedObject" mitgegeben wurde.
  Set NotesEmbeddedObject = _
  NotesRichTextItem.GetEmbeddedObject(stringName)

- **GetFormattedText**
  Der Inhalt eines Richtext-Feldes wird mit der Methode „GetFormattedText“ in Form von einfachem Text in eine Zeichenkette übernommen. Die Formatierung bezieht sich lediglich auf die Umwandlung von Tabulatorzeichen in die entsprechende Anzahl von Leerzeichen. Außerdem kann die gewünschte Zeilenlänge angegeben werden. Wird keine Zeilenlänge angegeben, beträgt der Standardwert 80 Zeichen.
  stringText = _
  NotesRichTextItem.GetFormattedText(booleanTabErsetzen, _
  integerZeilenlänge)

Programmtext

```
‘/// Aus einem vom Benutzer ausgewählten Dokument sollen
‘/// Kommentare entfernt und der Text als Textdatei gespeichert
‘/// werden.

Sub saveTextDatei(strTextDatei As String)
‘///
    ‘/// Workspace- und Session-Objekt erstellen
    Dim ws As New NotesUiWorkSpace
    Dim ses As New NotesSession

    ‘/// Dokumentensammlung definieren
    Dim col As NotesDocumentCollection

    ‘/// Dokument, Richtext und Textvariable definieren
    Dim doc As NotesDocument
    Dim rtiBody As NotesRichTextItem
    Dim strText As String
```

```
'/// Um die Zeilenschaltungen und Zeilenumbrüche zu
'/// entfernen, werden Variable definiert, die mit
'/// den entsprechenden Zeichen gefüllt werden.
Dim strCR_LF As String
Dim strLF As String

'/// Textvariable für den geänderten Text
Dim strAusgabeText As String
Dim bolKommentar As Integer
Dim strZeile As String

'/// Variable initialisieren
strCR_LF = Chr$(13) + Chr$(10)
strLF = Chr$(10)
bolKommentar = False

'/// Markiertes Dokument in die Dokumentensammlung
'/// aufnehmen. Es wird nur das erste markierte Dokument
'/// verarbeitet. Sollten alle Dokumente verwendet werden,
'/// müsste man eine entsprechende Schleife einfügen.
Set col = ses.CurrentDatabase.UnprocessedDocuments
Set doc = col.GetFirstDocument

'/// Richtext-Objekt für das Body-Feld erstellen
Set rtiBody = doc.GetFirstItem("Body")

'/// Text aus dem Body-Feld in eine Textvariable übernehmen.
'/// Tabulatorzeichen werden nicht ersetzt. Es wird eine Zeilen-
'/// länge von 120 Zeichen verwendet.
strText = rtiBody.GetFormattedText(False, 120)
```

```
'/// Alle CR-LF durch LF ersetzen
While Instr(1, strText, strCR_LF) > 0
    strText = Strleft(strText, strCR_LF) & _
    strLF & Strright(strText, strCR_LF)
Wend

'/// Ende auf LF prüfen, damit ein sauberes Ende erkannt wird.
If Right(strText, 1) <> strLF Then strText = strText & strLF

'/// Neuen Text ohne Kommentar bilden
While strText <> ""

    '/// Zeile bilden
    strZeile = Strleft(strText, strLF)

    '/// Text um die Zeile reduzieren
    strText = Strright(strText, strLF)

  '/// Prüfen, ob die Zeichenfolge %REM am Zeilenanfang steht.
    If Ucase(Left(strZeile, 4)) = "%REM" Then
        bolKommentar = True
    End If

    '/// Wenn das nicht der Fall ist, prüfen, ob eine REM-
    '/// Kommentarzeile vorliegt.
    If Not bolKommentar Then

        If Ucase(Left(strZeile, 4)) <> "REM " Then

        '/// auf Hochkomma und 3 Schrägstriche prüfen
        If Instr(1, strZeile, l'///l) > 0 Then
            strZeile = Strleft(strZeile, l'///l)
        End If
```

```
            '/// Leerzeilen nicht ausgeben, sondern nur Textzeilen in
            '/// die Ausgabe übernehmen und einen Zeilenwechsel
            '/// anhängen.
            If Trim(strZeile) <> "" Then _
                strAusgabeText = strAusgabeText & _
                strZeile & strCR_LF
            End If

        End If

        '/// Da %REM den Text solange als Kommentar
        '/// kennzeichnet, bis %ENDREM ihn aufhebt, wird über die
        '/// Variable "bolKommentar" die Textausgabe gesteuert.
        If Ucase(Left(strZeile, 8)) = "%END REM" Then
            bolKommentar = False
        End If

    Wend

    '/// Den fertigen Text in eine Datei abspeichern. Der Name der
    '/// Datei wird der Funktion übergeben.
    If strAusgabeText <> "" Then
        Open strTextDatei For Output As #1
        Width #1, 120
        Print #1, strAusgabeText
        Close #1

        '/// Fertigstellungsmeldung in der Statuszeile ausgeben.
        Print "Modul: "; strTextDatei; " erstellt"
    Else
        Print "Keine Datei erstellt!"
    End If
```

```
'///
End Sub
```

- **GetMIMEEntity (s. NotesItem)**

- **GetNotesFont (Release 6)**
  Ermittelt und setzt die Schriftart eines Richtext-Bereiches über den dazugehörigen NotesRichtextStyle.
  integerFont = NotesRichTextItem.GetNotesFont(stringName, _ booleanAnlegen)

- **GetUnformattedText (Release 6)**
  Bei der Methode „GetUnformattedText“ wird der Inhalt eines Richtext-Feldes als einfache Zeichenkette ermittelt. Tabulatorzeichen und Zeilenschaltungen bleiben unverändert erhalten.
  stringText = NotesRichTextItem.GetUnformattedText( )

- **GetValueDateTimeArray (s. NotesItem)**

- **New**
  Dient zur Erstellung eines neuen Objektes der Klasse NotesRichtextItem.
  Set NotesRichTextItem = New NotesRichTextItem(NotesDocument, stringName)

- **Remove (s. NotesItem)**

- **Update (Release 6)**
  Führt alle noch offenen Änderungen durch.
  Call notesRichTextItem.Update

6.3.7 Externe Daten und Programme

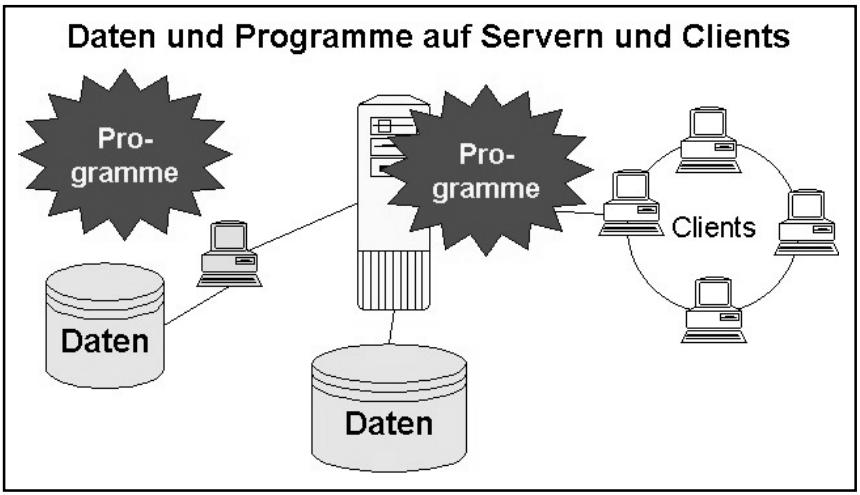
Solange die Daten und Programme innerhalb des Domino-Systems liegen, kann mit dessen Methoden gearbeitet werden. Schwieriger wird es, wenn man auf externe Daten zugreifen will oder externe Programme verwendet werden sollen. Die einfachste Methode externe Daten zu verarbeiten, besteht darin, sie zu importieren. Sollen die Daten aber an ihrem Ursprung bleiben und von dort aktuell gelesen oder sogar dort verändert werden, so ergeben sich die typischen Schnittstellenprobleme, die durch so genannte Treiber gelöst werden können. Lotus Domino kann Standardtreiber (wie z. B. ODBC) einsetzen, stellt aber auch eigene Schnittstellensysteme, die so genannten Connectoren zur Verfügung.

Externe Programme zu steuern oder zu verwenden ist nur mit Hilfe der Methoden des jeweiligen Betriebssystems möglich. Unter Windows kann Lotus Domino z. B. auf DDE und OLE zurückgreifen und damit Verbindungen zu anderen Programmen herstellen.

Februar 2001 / 4. EL

6.3.7.1 Externe Daten verarbeiten mit ODBC und Lotus Connectoren

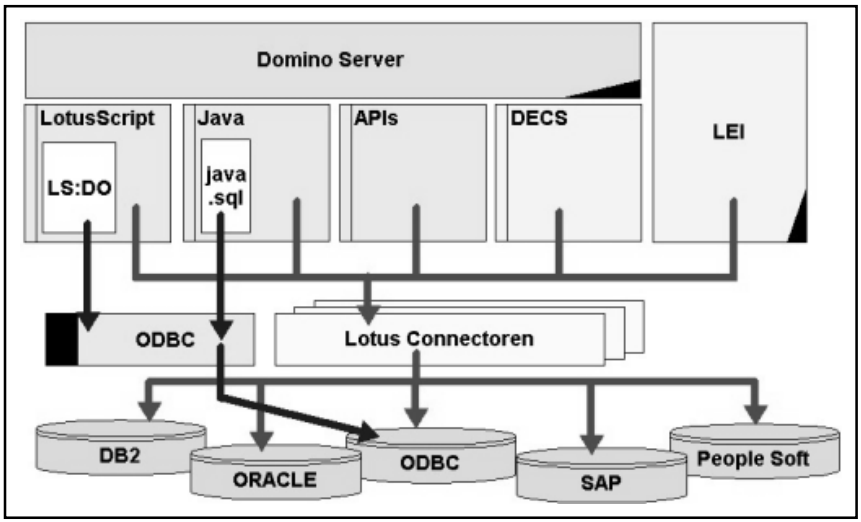
Es handelt sich bei Lotus Domino um ein System, das sowohl auf den Servern als auch auf den Clients Programme einsetzen und Daten vorhalten kann. Je nachdem, wo Programme und Daten angesiedelt sind, werden unterschiedliche Verfahren für den Datenzugriff verwendet.



Programme und Daten

Die Verfahren für den Datenzugriff hängen von der eingesetzten Programmiersprache und der Art der relationalen Datenbank ab. Als Standard unter Windows hat Microsoft den ODBC-Datenzugriff entwickelt. Parallel dazu bieten die Java-Systeme JDBC an, das auch auf Nicht-Windows-Systeme zugreifen kann. Im Bereich der C-Sprachen kommen noch Low-Level-Datenzugriffe

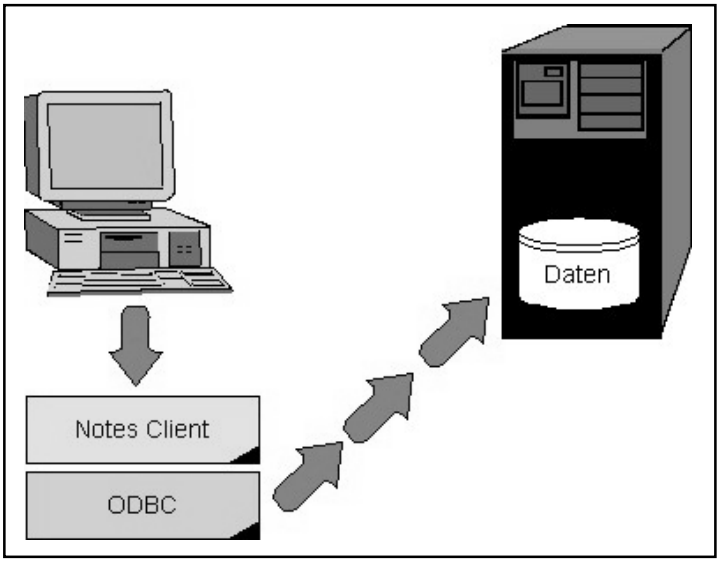
hinzu. Um dem Programmierer eine einheitliche Schnittstelle für den Zugriff auf externe Daten zu bieten, hat Lotus die sogenannten Connectoren geschaffen. Es handelt sich dabei um Betriebssystem- und Datenbank-abhängige Funktionssammlungen in Form von Hilfsprogrammen, die z. B. unter Windows als DLL-Dateien mit der Endung „.LCX“ ausgeführt sind. Die Lotus Connectoren greifen direkt auf die jeweiligen Datenbankschnittstellen zu.



Datenzugriff auf Datenbanken und ERP-Systeme

Werden Domino-Programme auf einem Notes-Client ausgeführt, so kann – unter Zuhilfenahme eines Treibers – direkt auf die externen Daten zugegriffen werden, sobald diese im Netz verfügbar und mit den entsprechenden Treibern versehen sind. Soll also z. B. eine Anwendung eine Adressliste anzeigen, deren Daten auf

einem iSeries 400-System liegen, so können diese Daten über einen ODBC-Treiber geholt und in einem Listenfeld angezeigt werden.



Zugriff eines Client-Programms auf externe Daten

Bei dieser Form des Datenzugriffs ist es möglich, die Daten über die Formelsprache, über LotusScript und über Java-Agenten zu verwenden.

Bei LotusScript können „LS:DO“ (LotusScript Data Objects) oder die Lotus Connectoren verwendet werden. Beide Systeme verwenden Klassen, die in Form von Programmbibliotheken zur Verfügung gestellt werden. Die Bibliotheken werden in dem Optionsbereich eines Programm-Moduls definiert.

Die Anweisung lautet:

uselsx „*LSXODBC“ für LS:DO und
uselsx „*LSXLC“ für einen Connector

Die Bearbeitung der Daten erfolgt bei ODBC über die Klassen:

- ODBCConnection
- ODBCQuery
- ODBCResult

Programmtext

Options:

```
‘// ODBC-Bibliothek einbinden
uselsx “*LSXODBC”
```

Declarations:

```
‘// Objekte definieren
Dim conAdr As ODBCConnection
Dim qryAdr As ODBCQuery
Dim rstAdr As ODBCResultSet
```

Postopen:

```
,// Verbindung einrichten
Sub Postopen(Source As Notesuidocument)
Set conAdr = New ODBCConnection
Set qryAdr = New ODBCQuery
Set rstAdr = New ODBCResultSet
Set qryAdr.Connection = conAdr
Set rstAdr.Query = qryAdr
```

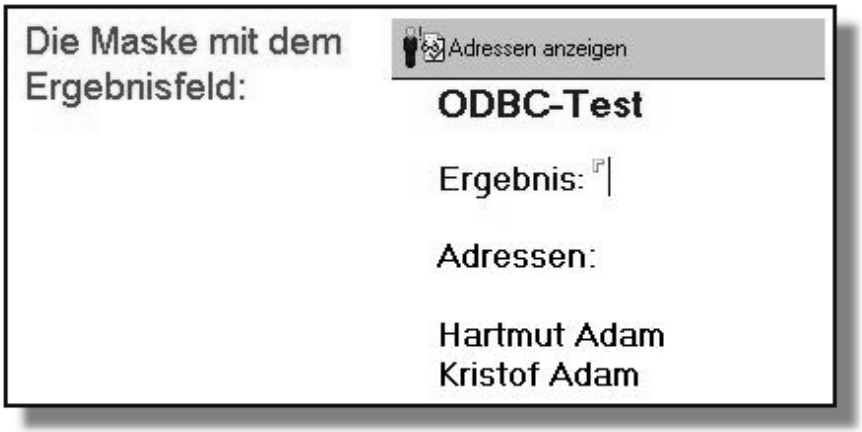
```
qryAdr.SQL = _
“SELECT * FROM MITGLIEDER WHERE “ & _
“PLZ = ‘74211’”
conAdr.ConnectTo(“Adressen”)
End Sub
```

Schaltfläche:

```
,// Namen über eine Schaltfläche anzeigen
Sub Click(Source As Button)
Dim ws As New NotesUiWorkspace
Dim uidoc As NotesUiDocument
Dim strVor As String
Dim strNach As String
Dim strMsg As String

Set uidoc = ws.CurrentDocument
rstAdr.Execute

If rstAdr.IsResultSetAvailable Then
    strMsg = “Adressen:” & Chr(10)

    Do
    rstAdr.NextRow
    strVor = rstAdr.GetValue(„Vorname“, strVor)
    strNach = rstAdr.GetValue(„Name“, strNach)
    strMsg = strMsg & strVor & „ „ & _
    strNach & Chr(10)
    Loop Until rstAdr.IsEndOfData

    uidoc.FieldSetText “Ergebnis”, strMsg
End If
End Sub
```

```
Queryclose:
,// Verbindung abbrechen
Sub Queryclose(Source As Notesuidocument, Continue As Variant)
conAdr.Disconnect
End Sub
```

In dem Beispiel wird das Ergebnis der Abfrage in einem Maskenfeld angezeigt. Es könnte auch ein Auswahlfeld gefüllt werden.



Maske mit Schaltfläche und Ergebnis der Abfrage

Über die Lotus Connectoren lauten die Klassen:

- LCSession
- LCConnection
- LCFieldList
- LCField

- LCStream
- LCDateTime
- LCNumeric
- LCCurrency

Der Vorteil der Verwendung von Lotus Connectoren besteht darin, dass die gleichen Klassen eingesetzt werden können, unabhängig davon, auf welcher Plattform das Programm letztendlich ausgeführt wird. Auch LEI (Lotus Enterprise Integrator) und DECS (Domino Enterprise Connection Services) arbeiten über die Connectoren. Neben den Standard-Connectoren sind solche für ERP-Systeme verfügbar. Diese “Premium-Connectoren” erlauben den Zugriff auf Daten von SAP, PeopleSoft u. a. und müssen zusätzlich gekauft werden.

Das Beispiel unter Verwendung der Connectoren-Klassen sieht wie folgt aus:

Programmtext

Options:
,// Connector einbinden
uselsx „*LSXLC“

Schaltfläche:
,// Einschließlich der Definitionen
Sub Click(Source As Button)
Dim ses As New LCSession
Dim conAdr As New LCConnection(“odbc2”)
Dim lcAdressListe As New LCFieldList

```
Dim lcVorname As LCField
Dim lcName As LCField
Dim count As Integer

Dim ws As New NotesUiWorkspace
Dim uidoc As NotesUiDocument
Dim strVor As String
Dim strNach As String
Dim msg As String

ses.ClearStatus
Set uidoc = ws.CurrentDocument
conAdr.Server = “Adressen”
conAdr.connect
conAdr.Metadata = „Mitglieder“
conAdr.Fieldnames = _
„adrvname, adrnname, adrplz“
conAdr.Select Nothing, 1, lcAdressListe

If lcAdressListe.Recordcount > 0 Then
Set lcVorname = _ lcAdressListe.LookUp(“adrvname”)
Set lcName = lcAdressListe.LookUp(“adrnname”)

    While conAdr.Fetch (lcAdressListe) > 0
Call uidoc.FieldAppendText(“AdressListe”, _ lcVorname.Text(0) & “
“ & _
lcName.Text(0) & Chr(10))
Wend

End If

End Sub
```

Der Zugriff auf externe Daten über den Server erfolgt entweder direkt über eigene Schnittstellen der Datenbank oder über Lotus Connectoren. Der Zugriff auf die DB2 UDB für AS/400 erfolgt mit Hilfe von Funktionen, LS:DO oder mit Hilfe der Connectoren über das SQL Call Level Interface (CLI) ohne ODBC-Treiber. Der Client benötigt in diesem Fall nur die Notes-Client-Software und die Netzwerkverbindung zum Server.

Als Beispiel für einen Zugriff auf eine physische Datei über den DB2-Connector wird ein Agent verwendet, der auf dem Server gestartet läuft. Dazu wird eine Schaltfläche im Client-Programm verwendet. Der Agent liest die physische Datei und gibt den Inhalt der Datensätze an der Konsole aus:

Programmtext

Die Schaltfläche:

```
Sub Click(Source As Button)
Dim Session As New Notessession
Dim Agent As Notesagent
Dim Currdb As Notesdatabase
Dim Rc As Integer

Set CurrDb = Session.Currentdatabase
Set agent = currdb.getagent("LC1")

Rc = agent.runonserver()
End Sub
```

Der Agent:
Options:
,// Connector einbinden
uselsx „*LSXLC“

Initialize:
Sub Initialize
Dim ses As New LCSession
Dim conAdr As New LCConnection(“db2”)
Dim Keys As New LCFieldList
Dim Key As LCField
Dim lcAdressListe As New LCFieldList
Dim lcVorname As LCField
Dim lcName As LCField
Dim count As Integer
Dim strVor As String
Dim strNach As String
Dim msg As String

ses.ClearStatus

, Muss angepasst werden !!!
conAdr.database = „Name der AS/400“
conAdr.userid = “Benutzer”
conAdr.password = „Kennwort des Benutzers“
conAdr.connect

, Muss angepasst werden !!!
conAdr.Metadata = „Bibliothek.Adressen“
conAdr.Fieldnames = „adrnname“
Set Key = Keys.Append(„Adrnname“, LCTYPE_TEXT)
Key.Text = “Daum”

```
Print „anf“
count = conAdr.Select (Nothing, 1, lcAdressListe)
Print “count “ & lcAdressListe.RecordCount

While conAdr.Fetch (lcAdressListe) > 0
Set lcName = _ lcAdressListe.LookUp(“Adrnname”)
Print “Erg “lcName.Text(0)
Wend

Print „ende“

End Sub
```

Das Ergebnis an der Domino-Konsole wird mit Hilfe des Befehls WRKDOMCSL und der Auswahl 8 bei dem Server angezeigt. Es ergibt sich das folgende Bild:

```
01:45   Addin: Agent printing: anf
01:51   Addin: Agent printing: count 1
01:51   Addin: Agent printing: Erg Feldbusch
01:51   Addin: Agent printing: Erg Daum
01:51   Addin: Agent printing: Erg Beckenbauer
01:51   Addin: Agent printing: ende

resh   F6=Print
ttom   F21=Command line
```

Ausgabe der Abfrage an der Domino-Konsole

In der Praxis wird natürlich das Ergebnis nicht über die Konsole, sondern über ein Parameterdokument an den aufrufenden Agenten zurückgegeben.

Diese Seite ist zur Zeit nicht belegt.

6.4 Javascript

Obwohl der Name es nahe legt, ist Javascript völlig unabhängig von Java. Javascript wird während der Ausführung interpretiert und wurde als Erweiterung zu HTML für den Einsatz in Browsern entwickelt. Lotus Notes und Domino werden immer stärker als Browser und Internet-Server positioniert. Im Zuge dieser Entwicklung wurde auch Javascript übernommen und als Erweiterung der Gestaltungsobjekte genutzt. Die Integration ist zur Zeit noch nicht abgeschlossen. Viele Ereignisse können nur in Browsern, aber nicht im Notes-Client genutzt werden. Es ist jedoch absehbar, dass Javascript eine wichtige Rolle in der Anwendungsentwicklung unter Lotus Domino spielen wird. Je mehr Notes-Anwendungen auch über Browser benutzbar sein sollen, desto mehr werden Browser-orientierte Sprachen wie Javascript zum Einsatz kommen.

Diese Seite ist zur Zeit nicht belegt.

6.5 Java

Java hat sich in den letzten Jahren nicht nur im privaten Bereich, sondern vermehrt auch im kommerziellen Bereich mehr und mehr etablieren können. Neben den in Notes integrierten Programmiersprachen LotusScript und der Formula-Sprache wurde die Implementierung von Java in Notes wesentlich erweitert und verbessert, so dass man zwischenzeitlich Java als Programmiersprache im Notes-Umfeld als Alternative für die übrigen Anwendungsentwicklungs-Möglichkeiten ansehen kann. Gerade auch im Hinblick auf die anstehenden Neuerungen im Notes/Domino-7-Umfeld dürfte Java in Zukunft eine stärkere Bedeutung im Entwicklungsumfeld finden, als dies heute der Fall ist – denn neben den seit Jahren fest etablierten Entwicklungs-Sprachen LotusScript und der gleichfalls weit verbreiteten Formel-Sprache – sind Java-Programme im Domino/Notes-Umfeld noch wenig etabliert.

Ein weiterer Grund für die steigende Akzeptanz von Java im Notes-Umfeld ist, dass die Zahl der Java-Programmierer ungleich größer ist als die der Lotus-Script oder @Formel-Sprachen-Entwickler, so dass in Zukunft wohl auch deswegen die Bedeutung von Java im Notes-Umfeld noch zunehmen wird. Betrachtet man zudem die Tendenzen der IBM, Lotus(Domino) und die WebSphere-Familie miteinander zu verbinden, dann liegt der Schluss nahe, dass Java im Domino-Umfeld auch von dieser Seite in Zukunft weiter gefördert wird. Alles in allem sicher kein Grund, sich von den bewährten Entwicklungssprachen LotusScript und Formelsprache zu trennen, sicher aber ein Grund, der Anwendungsentwicklung Domino/Notes mit Java eine gewisse Aufmerksamkeit zu widmen.

In diesem Kapitel wollen wir die Anwendungsentwicklung mit Java in einem Notes/Domino-Umfeld näher betrachten. Dabei setzen wir zwar Grundkenntnisse in der Programmiersprache Java voraus – denn die Vermittlung dieser Grundkenntnisse würde den Rahmen dieses Kapitels bei weitem sprengen – zudem gibt es auf dem Markt eine Fülle an guter Literatur, die die Grundkenntnisse vermittelt – gleichwohl wollen wir schrittweise die Einführung in die Spezifika der Java-Anwendungsentwicklung in Verbindung mit Domino/Notes erläutern – so dass auch Java-Anfänger verständlich an dieses Thema herangeführt werden. Aber auch Java-Kenner dürften in diesem Kapitel einige wichtige Informationen im Umgang mit Java und Notes erhalten.

6.5.1 Java-Agenten

Die Integration von Java in Notes erfolgt in Form von Java-Agenten. Agenten kennen wir bereits aus anderen Bereichen der Anwendungsentwicklung im Notes-Umfeld, wie zum Beispiel:

- LotusScript-Agenten
- Einfache Agenten
- Formel-Agenten
- Java-Agenten
- Java-Script-Agenten

Interessant bei der Verwendung von Java-Agenten ist, dass der Entwickler entscheiden kann, ob der Java-Code innerhalb des Notes-Designers entwickelt werden soll, oder ob ein anderes Kodierungstool für die Java-Anwendungsentwicklung verwendet und der fertige Java-Code nach der Kodierung in Notes importiert wird.

In diesem und den nachfolgenden Beispielen verwenden wir für die Java-Programmierung ausschließlich den Domino-Designer.

Schauen wir uns zunächst einmal die Entwicklung eines einfachen Java-Agenten innerhalb des Domino-Designers an:

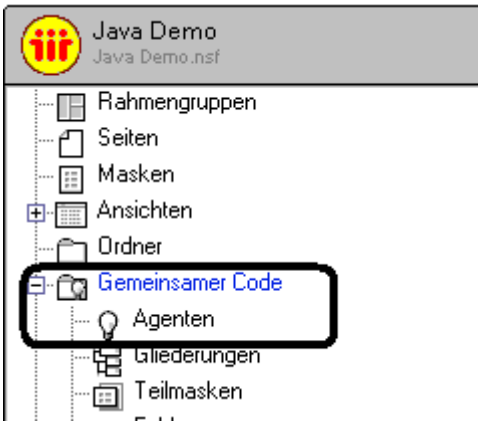
Für die Verdeutlichung der Beispiele in diesem Kapitel verwenden wir zunächst eine Beispiel-Datenbank „Java Demo“, die wir zunächst in dem Domino-Designer ohne Verwendung bestimmter Datenbankvorlagen als leere Datenbank erstellen.

November 2004 / 19. EL

Damit wir nun unseren ersten Java-Agenten innerhalb von Notes schreiben können, gehen wir wie folgt vor:

❶❷❸ Ausführungsschritte:

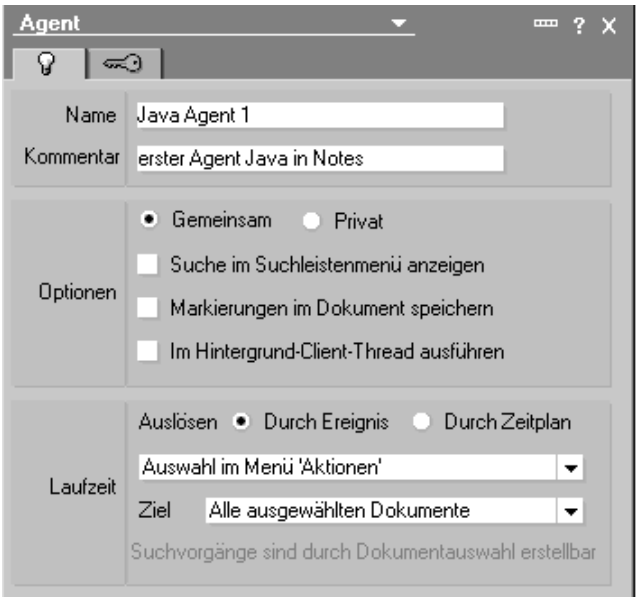
- Öffnen Sie die neue Datenbank „Java Demo“ oder erstellen Sie diese im Domino-Designer über „Datei / Datenbank / Neu“ (verwenden Sie keine Vorlage, sondern erstellen Sie eine leere Datenbank)
- Im Domino Designer erweitern Sie den Eintrag „Gemeinsamer Code“ im linken Fensterbereich für die neue Datenbank „Java Demo“ durch einen Klick auf das „+“-Zeichen.
- Wählen Sie den Eintrag „Agenten“ aus.



Agenten öffnen

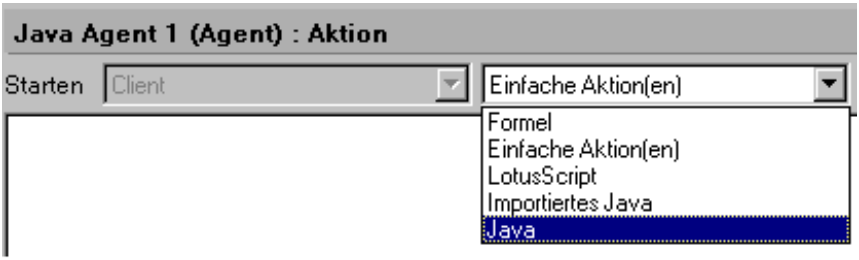
- Im rechten Fensterabschnitt öffnet sich nun ein leeres Arbeitsfeld, in dessen oberem Bereich unter anderem die Schaltfläche „Neuer Agent“ zu finden ist. Diese wählen wir nun aus.

- Jetzt können wir in dem Eigenschaftsfenster dem Agenten einen Namen geben. Dieser Name sollte möglichst aussagekräftig sein und den Inhalt des Agenten widerspiegeln. In unserem Beispiel verwenden wir „Java Agent 1“ als Agentenname. In dem Feld „Kommentar“ kann eine beschreibende Information für diesen Agenten eingegeben werden, die im Gegensatz zu dem Namen keine programmrelevante Funktion darstellt. Die übrigen Parameter in diesem Eigenschaftsfenster sollten bereits bekannt sein, denn hier wird nicht zwischen den unterschiedlichen Programmiersprachen unterschieden, die später innerhalb des Agenten angewendet werden können. Deshalb verzichten wir an dieser Stelle auf eine nähere Erläuterung dieser Parameter und verweisen auf die entsprechenden Kapitel in diesem Buch.



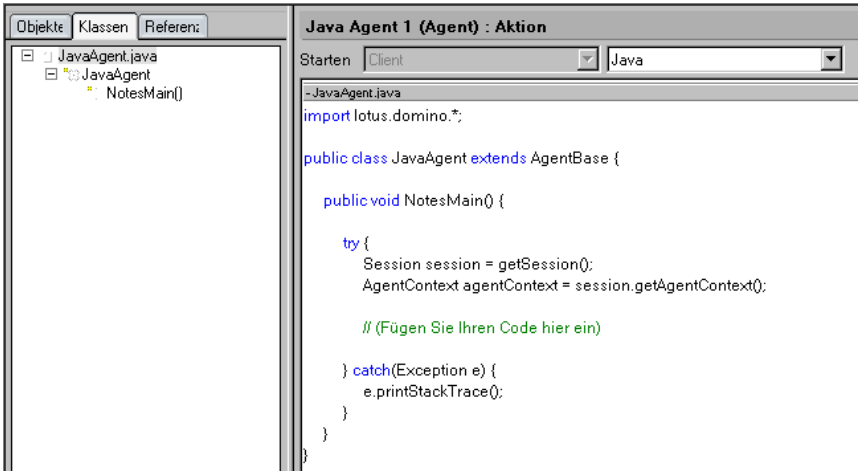
Agent-Eigenschaften

- Nachdem der Name und wahlweise der Kommentar eingegeben worden sind, kann das Eigenschaftsfenster durch einen Klick auf das in der oberen rechten Ecke befindliche „X“ geschlossen werden.
- Im oberen rechten Bereich der Bildschirmseite befindet sich ein Auswahlfenster für die in diesem Agenten anzuwendende Programmiersprache. Dies können sein:
  - Formel
  - Einfache Aktion(en)
  - Lotus Script
  - Importiertes Java
  - Java



Programmierumgebung auswählen

- In der Liste der Programmiersprachen wählen wir nun den Eintrag „Java“ aus.
- Nach der Auswahl „Java“ wird in dem Editierbereich ein Grundgerüst eines Java-Codes geöffnet.



Java-Grundgerüst

➢ Um dem Einstiegsentwickler das Leben ein wenig zu vereinfachen, wird in dem Gerüst die Stelle markiert, wo der individuelle Code platziert werden kann.

Mit diesen Schritten sind nun die ersten Vorbereitungen für die Anwendungsentwicklung von Java in Notes getroffen.

Betrachten wir das Grundgerüst ein wenig näher. Dazu schauen wir uns links neben dem Editierbereich die Anzeigenspalte an. Diese enthält einige Tabellen – unter anderem auch die Tabelle „Klassen“. Mit dem Grundgerüst wurde die erste Java-Klasse „JavaAgent“ erstellt.

Klasse JavaAgent

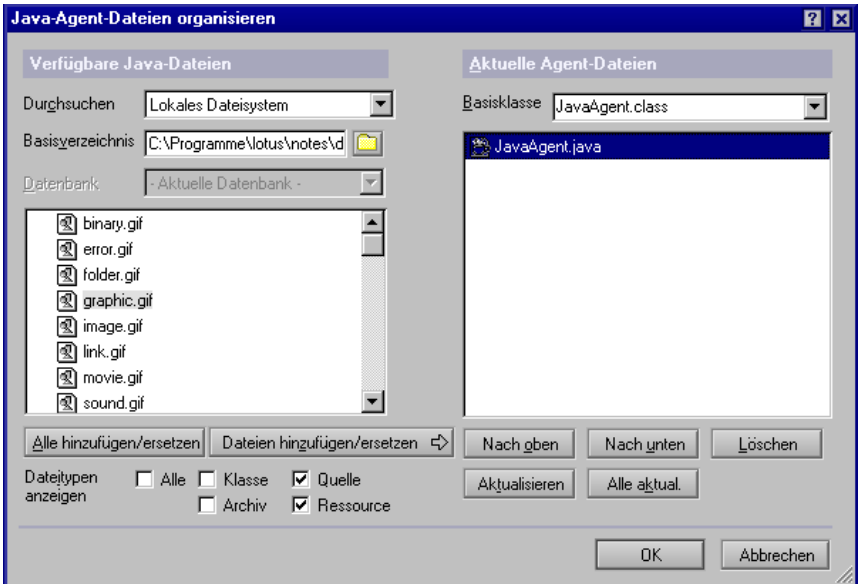
Die Anwendungsentwicklung im Umfeld von Java basiert im Wesentlichen auf der Verwendung der Klassen. Es ist selbstredend, dass neben der Basis-Klasse „JavaAgent“ weitere Klassen in diesem Agenten erstellt werden können. Neue Klassen können einfach durch einen Klick auf die Schaltfläche „Neue Klasse“ – diese finden wir im unteren Bereich des Fensters – generiert werden.

Neue Klasse

Jede Klasse kann für sich entsprechend angepasst werden. Dazu dient die Schaltfläche „Projekt bearbeiten“, die sich ebenfalls neben der Schaltfläche „Neue Klasse“ im unteren Bereich der Fensteranzeige befindet.

Projekt bearbeiten

Ein Projekt kann in dem neuen Fenster bearbeitet werden.



Java-Agent-Dateien organisieren

Was es mit den unterschiedlichen Angaben in diesem Bereich auf sich hat, werden wir später in diesem Kapitel näher betrachten. Besondere Vorsicht ist mit der Option „Löschen“ geboten – denn durch einen Klick kann die gesamte Arbeit innerhalb einer Klasse verloren gehen!

6.5.1.1 Basisklasse

Jeder Java-Agent enthält zunächst eine Basisklasse, die die Methoden enthält, die während des Starts des Agenten ausgeführt werden. Jede Basisklasse wird von der Meta-Klasse „*AgentBase*“ abgeleitet.

In folgenden Beispiel ist dies mit dem Code-Fragment Meta-Klasse der Fall:

```
public class JavaAgent extends AgentBase
```

Meta-Klasse

Einer der wesentlichen Bestandteile dieser Basisklasse ist die Methode „*NotesMain*“. Der Inhalt dieser Methode wird initial bei jedem Start des Agenten ausgeführt

```
public void NotesMain()
```

NotesMain

Die Klasse „AgentBase“ stellt, wie bereits weiter oben beschrieben, die „Mutter“ für die Basisklasse dar. In der Klassenhierarchie

ist dieser Klasse die Klasse „Agent“ übergeordnet. Die Klasse „Agent“ enthält wichtige Informationen und Eigenschaften für die Verarbeitung von Agenten.

Betrachten wir nun die möglichen Eigenschaften eines Agenten:

Jeder Agent kann:

- öffentlich oder
- privat sein

Darüber hinaus kann ein Agent

- aktiv oder
- inaktiv sein

Diese Eigenschaften werden unter anderem in der Klasse „Agent“ dargestellt:

Programmtext

```
class Agent {

  boolean isEnabled();
  void setEnabled ( boolean flag );

  boolean isPublic ();
  boolean isNotesAgent ();
  boolean isWebAgent();
}
```

Gehen wir eine Ebene tiefer in den Eigenschaften einer Klasse – hier befinden sich unter anderem Informationen wie zum Beispiel:

- Name des Autoren des Agenten
- Letzte Ausführung
- Kommentare
- Server Name
- Trigger

Schauen wir uns dies einmal in dem folgenden Code-Beispiel genauer an:
Basis für dieses Beispiel ist der noch leere Agent, den wir zuvor erstellt haben. Dieser verfügt derzeit nur über das Basisgerüst.

Wir definieren erst eine Klasse „Beispiel1“ und geben anschließend die Eigenschaften des Agenten aus, die anzeigen, wer diesen erstellt hat und wer der Eigner des Agenten ist. Zudem wird der Kommentar des Agenten angezeigt und das Datum der letzten Ausführung des Agenten:

Programmtext

Java Agent 1 (Agent) : Aktion

Starten | Client | Java

- Beispiel1.java

```
import lotus.domino.*;

public class Beispiel1 extends AgentBase {
public void NotesMain() {
    try {
        Session se = getSession();
        AgentContext ac = se.getAgentContext();
        Agent ag = ac.getCurrentAgent();
        Database db = ac.getCurrentDatabase();

        System.out.println ( "Name Agent:" + ag.getName() );
        System.out.println ( "Eigner:" +
            ag.getCommonOwner() );
        System.out.println ( "Eigner:" + ag.getOwner() );
        System.out.println("Kommentar: " + ag.getComment() );
        System.out.println("zuletzt genutzt: " + ag.getLastRun() );


    } catch(Exception e) {
        e.printStackTrace();
        };
    };
};
```

Agent – Beispiel 1

Dieses kleine Beispiel soll nur grob den Aufbau der Klasse und dieses Programms darstellen.

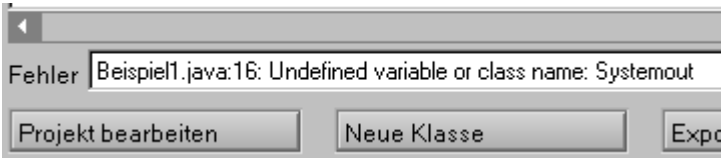
6.5.1.2 Kompilierung

Damit wir diesen Agenten verwenden können, sollte er zunächst kompiliert werden. Dazu klicken wir im unteren Bereich auf die Auswahl „Kompilieren“

Kompilieren

Achten Sie nach dem Kompilieren auf das Ergebnis dieses Umsetzungslaufes – sowohl Fehler als auch die Meldung einer erfolgreichen Kompilierung werden in der Fehlerleiste angezeigt.

Sollte der Sourcecode Syntaxfehler aufweisen, so werden diese mit einem Verweis auf die fehlerhafte Zeile angezeigt.

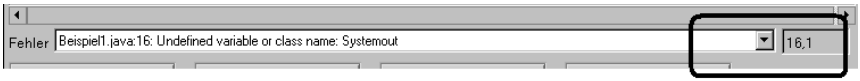


Kompilierfehler

In dieser Abbildung haben wir einen Fehler in der Source-Zeile 16. Hier wurde die Anweisung „systemout“ nicht gefunden. Dies hätte korrekt „system.out“ heißen müssen. Vor der Ausführung und Speicherung des Agenten müssen alle Fehler behoben sein. Mit dieser Fehlermeldung wird eine kleine Schwäche der Anwendungsentwicklung von Java in Notes deutlich:

In der Fehlermeldung wird direkt auf die fehlerhafte Source-Zeile verwiesen. („Beispiel1 .java:16“). In unserem Beispiel enthält die Zeile 16 einen Source-Fehler. Schön wäre in diesem Fall eine Nummerierung der Zeilen direkt in der Source-Ansicht oder eine farbliche Hervorhebung der fehlerhaften Zeilen – leider ist dies nicht der Fall – aber Lotus bietet wenigstens eine kleine Hilfe:

Im unteren Bereich befindet sich rechts neben der „Fehlerzeile“ die Anzeige der aktuellen Position in dem oberen Source-Editierbereich:



Cursor-Position

Nachdem alle Fehler in der Source behoben sind, kompilieren wir den Source-Code erneut.

Die „Fehlermeldung“ der abschließenden Kompilierung sollte dann so aussehen:

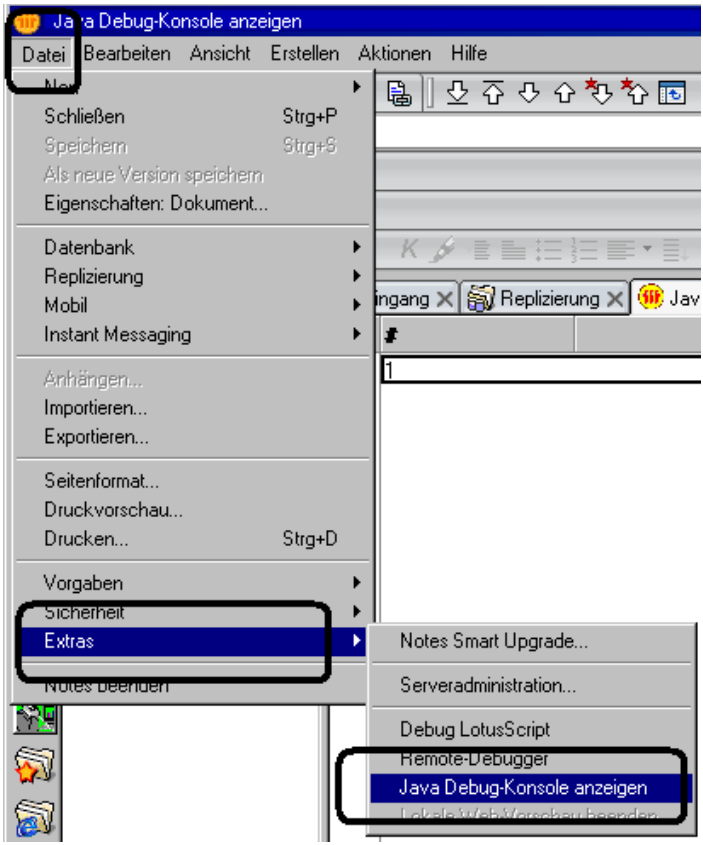
Kompilier-Ergebnis

Testen wir nun diesen Agenten. Dazu wechseln wir in den Notes Client und öffnen dort die neue Datenbank „Java Demo“, wenn diese nicht bereits geöffnet ist.

6.5.1.3 Java Debugger

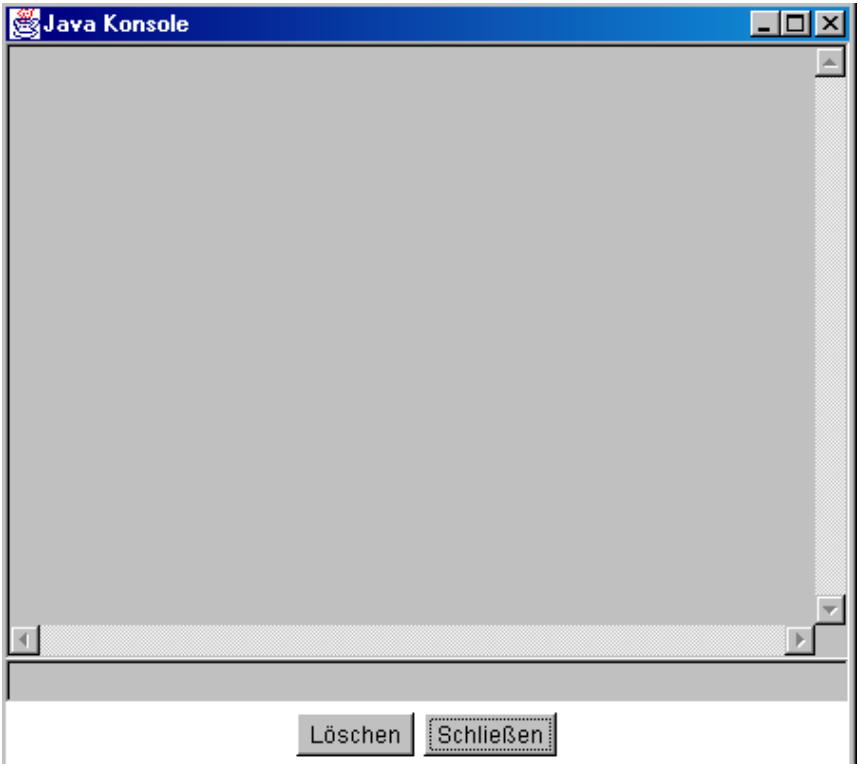
Um unseren ersten Agenten zu testen, verwenden wir den Java-Debugger, der in Lotus integriert ist. Dieser Debugger ist die Basis für die Prüfungen der Java-Anwendungsentwicklungen. Deshalb macht es bereits jetzt Sinn, auf diesen einzugehen.

Wie der Lotus-Script Debugger wird auch der Debugger für den Java-Bereich über die Auswahl „Datei / Extras“ gestartet.



Java Debugger

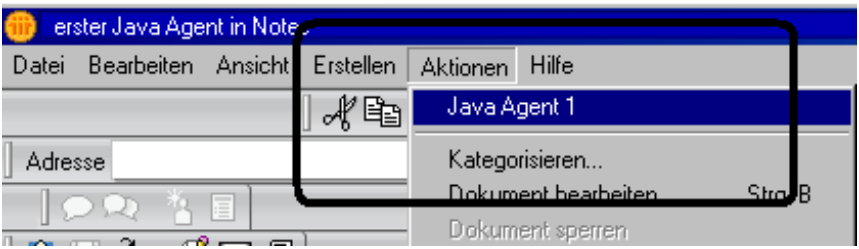
Wählen Sie den Eintrag „Java-Debug-Konsole anzeigen“ aus. Daraufhin wird die Java-Debug-Konsole in einem separaten Fenster geöffnet.



Java Debug-Konsole

Die Anzeige ist zunächst noch leer. Sobald wir ein Java-Programm – in unserem Beispiel den ersten Agenten – ausführen, wird in dieser Konsole der Debugger mitlaufen und die Ergebnisse unserer Programme anzeigen.

Also starten wir unseren Agenten.
Dazu wählen wir in der neuen Notes-Datenbank „Java Demo“ in der oberen Auswahlleiste des Notes-Clients „Aktionen / Java Agent1“ aus.



Agent starten

Sehen Sie was passiert? Zunächst einmal nichts …

Wir haben zuvor gesehen, dass die Java-Debug-Konsole ein eigenständiges Fenster ist. In dieses eigenständige Fenster wechseln wir in unserer Anzeige:

Ergebnis in Java-Konsole

Wir sehen erst jetzt die Ergebnisse des Agenten.

Wie man an diesem simplen Beispiel erkennen kann:
Eine elegante Debug-Funktion, wie wir sie zum Beispiel aus dem Bereich von LotusScript kennen, ist die Java-Debug-Konsole nicht, denn wir können nicht ohne Weiteres im direkten Programmablauf mit diesem Debugger in das Programm bzw. den Programmcode eingreifen.

Diese Seite ist zur Zeit nicht belegt.

6.5.2 Notes-Anwendungen und Java

Die Java-Anwendungsentwicklung im Umfeld von Domino/Notes bietet einige Besonderheiten, die besondere Beachtung benötigen.

Im Gegensatz zu den bisher bekannten Entwicklungswerkzeugen und Sprachen, die im Umfeld der Notes-Anwendungsentwicklung zum Einsatz kommen, bietet die Anwendungsentwicklung mit Java einige Neuerungen und auch Möglichkeiten.

Eine dieser Besonderheiten ist die Möglichkeit, Java-Programme im Notes-Umfeld auch ohne die Verwendung des Domino-Designers zu entwickeln. Das heißt, dass Java-Programme mit herkömmlichen Java-Editoren codiert werden können. Solche Programmcodes können bei Bedarf in den Domino-Designer importiert werden. Selbstverständlich kann die Codierung auch direkt im Domino-Designer erfolgen. Es ist sicher Geschmackssache, ob von dieser Möglichkeit Gebrauch gemacht wird.

In den Beispielen in diesem Kapitel wurde die Anwendungsentwicklung innerhalb des Domino-Designers vorgenommen. Vorteil bei diesem Verfahren ist, dass sämtlicher Programm-Code direkt der Notes-Datenbank zugeordnet ist.

Basis für die Java-Anwendungsentwicklung sind die Standards im Java-Umfeld, die bei der Integration von Java in Notes implementiert worden sind. Es gibt zwar eine Reihe von Besonderheiten bei der Java-Anwendungsentwicklung für/mit Domino, die grundsätzliche Struktur und die Elemente der Java-Programme, die für Notes entwickelt werden, unterscheiden sich jedoch nicht

von denen üblicher Java-Source-Codes und sind somit für Java-Kenner leicht zu lesen und entsprechend anzupassen.

Es ist nicht nur die Besonderheit, dass der Java-Programm-Code außerhalb des Domino-Designers entwickelt werden kann, sondern auch die Tatsache, dass die Java-Anwendungen, die Domino-Informationen verarbeiten, nicht innerhalb von Notes/Domino-Umgebungen ausgeführt werden müssen, sondern dass diese Anwendungen z. B. auch ohne einen Notes Client oder einen Domino Server ausgeführt werden können.

Dabei ist dann lediglich zu beachten, dass bestimmte Voraussetzungen erfüllt sein müssen, damit auf die Notes/Domino-Informationen zugegriffen werden kann.

Über Vor- und Nachteile lässt sich sicher streiten – wir nehmen diese Tatsache an dieser Stelle zunächst hin – denn für die Einführung in die Notes-/Java Anwendungsentwicklung genügt die Kenntnis dieser Möglichkeit.

Eines der wesentlichen Merkmale der Java-Anwendungsentwicklung liegt in der Verwendung bestehender Programm-Code-Fragmente, die zusammengesetzt und ergänzt in vielen Bereichen eine schnelle und effektive Anwendungsentwicklung ermöglichen. Solche Programm-Code-Fragmente finden wir auch speziell für Java<>Domino.

Die Anwendungsentwicklung von Java innerhalb des Domino-Designers wird innerhalb der Agenten ausgeführt.

6.5.2.1 Notes Standalone-Anwendung

Bei der Ausführung von Notes/Domino-Programmen sind bestimmte Besonderheiten in Bezug auf die Domino/Notes-Umgebung erforderlich. So muss vor dem Arbeiten mit einer Datenbank zum Beispiel sichergestellt sein, dass der Anwender über ausreichende Rechte für einen Zugriff auf die Informationen verfügt. Solche Bedingungen müssen auch von Java-Programmen geprüft werden, die außerhalb von Notes ausgeführt werden sollen.

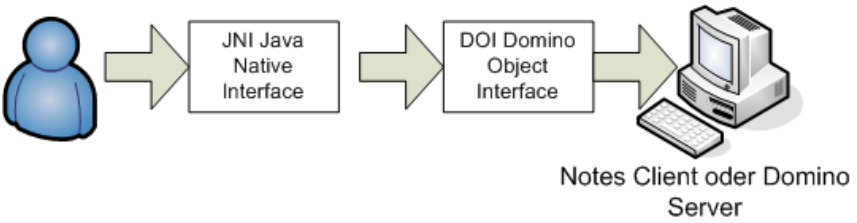
Wichtig sind in diesem Zusammenhang u. a. alle Informationen, die eine Notes-Sitzung enthält. Diese müssen in einem Standalone-Java-Programm – also einem Programm, dass außerhalb einer Notes/Domino-Anwendung ausgeführt wird, quasi simuliert werden. Dies kann durch einige wenige Code-Zeilen geschehen.

Bei der Ausführung solcher Standalone-Anwendungen wird unterschieden, ob das Programm auf einem Rechner ausgeführt wird, auf dem ein Notes Client oder ein Domino Server installiert ist, oder ob der Rechner über keine Notes/Domino-Installation verfügt.

Basis für die Ausführung solcher Standalone-Programme ist die Verwendung spezieller Schnittstellen innerhalb von Notes, die die Ausführung von Java-Programmen oder besser den Zugriff auf Notes-Daten über Java-Programme ermöglichen.

In diesem Zusammenhang können die „Java Native Interface Schnittstelle“ (kurz JNI) und die “Domino Object Interface Klassen” (kurz DOI) genannt werden.

DOI steht nicht allein für die Nutzung von Java in Notes zur Verfügung, sondern bildet zum Beispiel auch im Bereich von XML und Notes einen wichtigen Baustein.



Ausführung auf einem Rechner

Über diese Standardschnittstellen und Standardklassen ist eine direkte Kommunikation zwischen externen Tools mit Notes/Domino möglich.

Einen solchen Zugriff sehen wir uns im nächsten Beispiel näher an. In diesem Beispiel greifen wir mit einem kleinen Java-Beispielprogramm mit einigen Java-Domino-Klassen auf Domino-spezifische Informationen zu.

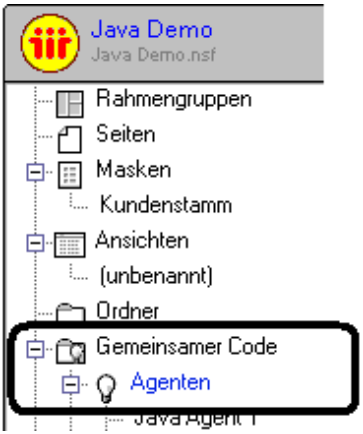
Voraussetzung für das Funktionieren dieser kleinen Beispielanwendung ist, dass auf dem Rechner, auf dem dieses Programm ausgeführt wird, ein Notes Client installiert ist, denn in dem Notes-Client sind u. a. auch die erforderlichen Java-Klassen enthalten, die wir in den Beispielen einsetzen und die eine Voraussetzung für die Kommunikation Java<>Domino darstellen.

6.5.2.1.1 Zugriff mit der Klasse „*NotesThread*“

Es gibt zwei unterschiedliche Möglichkeiten, um ein Java-Notes-Programm auf einem Rechner mit einer Notes-Installation auszuführen. Eine dieser Möglichkeiten ist die Verwendung der Klasse „*NotesThread*“, die ein Bestandteil des Package „*lotus.domino*“ ist. Die Klasse „*NotesThread*“ ist abgeleitet von der Klasse „*Thread*“**.**

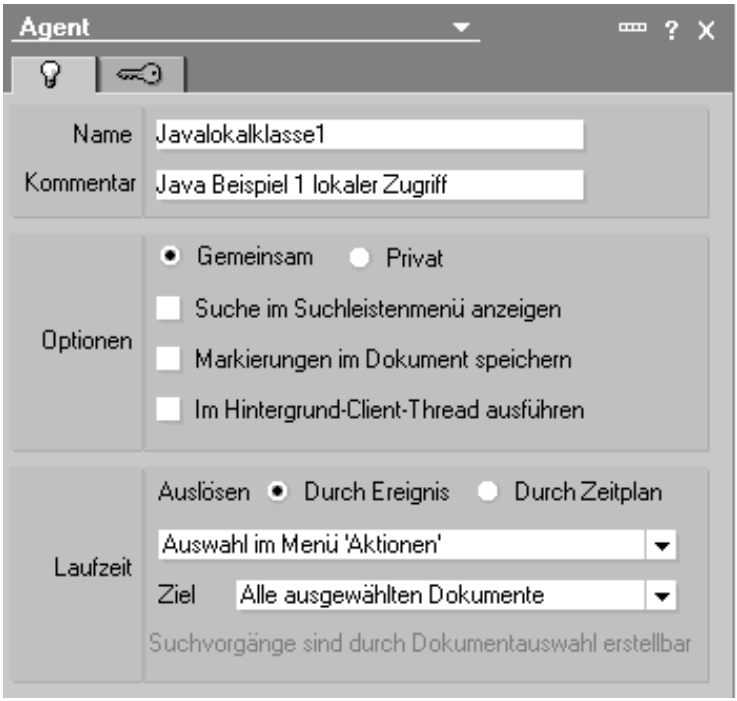
❶❷❸ Ausführungsschritte:

- Wir erstellen zunächst einen neuen Agenten im Domino Designer.
- Dazu öffnen wir die Datenbank „Java Demo“ – wenn diese nicht bereits geöffnet ist. Sollte diese Datenbank nicht existieren, so erstellen Sie diese bitte als leere Datenbank.
- Im linken Fensterbereich erweitern wir den Eintrag „Gemeinsamer Code“.
- Anschließend wählen wir den Eintrag „Agenten“ aus.



Agent erstellen

- Über dem sich nun öffnenden Editierfenster im rechten Bereich klicken wir auf den Eintrag „Neuer Agent“.
- In dem Fenster „Agent“ geben wir als Name „Javalokalklasse1“ ein.



Agent-Eigenschaften

➢ Nun folgt der eigentliche Source-Code:

Javalokalklasse1 (Agent) : Aktion

Starten | Client | Java

- Javalokalklasse1.java

```
import lotus.domino.*;  ❶

public class Javalokalklasse1 extends NotesThread {

   public static void Main( String argv[] ) {

   Javalokalklasse1 anwendung = new Javalokalklasse1();
   anwendung.start();
   };  ❷
   public void runNotes() {

      try {
         Session session = NotesFactory.createSession();

         // Dieser Code wird ausgeführt


      } catch(Exception e) {
         e.printStackTrace();
      }
   }
}
```

Javalokalklasse1

❶ ❷ ❸

Schauen wir uns nun dieses kleine Programm im Detail an:

- Bei der Angabe „*lotus.domino*“ handelt es sich um das Package, das unter anderem auch die Klasse „*NotesThead*“ enthält.
- In der Klasse „*NotesThreads*“ befindet sich u. a. auch die Methode „*runNotes*“, in die der auszuführende Programmcode gefügt wird.

Wir merken uns also für die Ausführung von Notes-Zugriffen mit einem Java-Programm, das außerhalb von Notes ausgeführt wird:

- Verwendung der Klasse „*NotesThreads*“
- Verwendung der Methode „*runNotes*“

6.5.2.1.2 Zugriff mit der Methode „*sinitThread*“

Die zweite Möglichkeit der Ausführung von Java-Notes-Programmen außerhalb von Notes – aber auf einem Rechner mit installiertem Notes Client oder Domino Server – bietet die Methode „*sinitThread*“, die anstelle der Klasse „*NotesThread*“ verwendet werden kann. „Anstelle der Klasse“ ist nur indirekt richtig, denn die Methode „*sinitThread*“ befindet sich in der Klasse „*NotesThread*“ – wir nutzen also ein Element in einer anderen Ebene – nicht in einer Java-Klasse, sondern in einer Methode.

Der Source-Code für einen solches Programm könnte (bewusst einfach gehalten!) wie folgt aussehen:

Programmtext

Javalokalklasse2 (Agent) : Aktion

Starten | Client | Java

- Javalokalklasse2.java

```
import lotus.domino.*;

public class Javalokalklasse2 {

    public static void Main( String argv[] ) {

        try {
        NotesThread.sinitThread();

            Session session = NotesFactory.createSession();

            // Dieser Code wird ausgeführt

        } catch(Exception ex) {
            ex.printStackTrace();
        } finally {
            NotesThread.stermThread();
    };
  };
};
```

Javalokalklasse2

Neben der Angabe der Methode „*sinitThread*“ benötigen wir noch eine weitere Methode für die Generierung der Notes-Umgebung: Die Methode „*createSession*“. Wie der Name dieser Methode bereits sagt, wird damit eine Notes-Session erstellt bzw. werden die erforderlichen Informationen, die eine Notes-Session enthält, mit dieser Methode zur Verfügung gestellt.

Mit diesen beiden Beispielen haben wir uns nun die Möglichkeiten angesehen, mit denen wir eine Notes-Umgebung quasi „simulieren“ können. Diese Informationen sollten in Java-Programmen verwendet werden, die auch außerhalb von Notes ausgeführt werden können und stellen die Basis für weiteren Programm-Code und entsprechende Zugriffe auf Notes-Informationen dar.

6.5.3 Notes-Datenbanken und Java

Die Betrachtung von Java in Notes-Umgebungen kann aus verschiedenen Perspektiven erfolgen. In der Praxis und für die ersten Schritte in dieser Entwicklungsumgebung bietet sich die Einführung nach den einzelnen elementaren Elementen von Notes an.

Deshalb beginnen wir zunächst bei der Betrachtung von Notes-Datenbanken und wie diese mit Hilfe von Java-Programmen erstellt, verwendet und bearbeitet werden können.

Lotus stellt für die Arbeit mit Datenbank unter Verwendung von Java eine eigene Klasse „*Database*“ zur Verfügung. Diese Klasse stellt die Basis für alle weiteren Behandlungen von Notes-Datenbanken mit Java dar.

Bevor wir mit einer Notes-Datenbank arbeiten, muss zunächst ermittelt oder angegeben werden, wo sich die Datenbank befindet. Dabei können unterschiedliche Fälle berücksichtigt werden:

- Notes-Datenbank in einem Verzeichnis auf einem lokalen Rechner
- Notes-Datenbank in einem Verzeichnis auf einem fernen Rechner

Bei der Betrachtung der Lokation ist es unerheblich, ob die Datenbank auf einem Notes-Client oder einem Domino-Server gespeichert ist.

Für die Ermittlung des Speicherortes einer Datenbank steht die Klasse „*DbDirectory*“ zur Verfügung. Diese befindet sich in der Klasse „*Session*“.

Das nachfolgende Code-Beispiel zeigt, wie mit einem Java Programm alle Notes-Datenbanken auf einem Rechner ermittelt werden können.

Dabei wird neben den bereits bekannten Java-Source-Code-Elementen unter anderem auch eine Schleifenprogrammierung verwendet, mit der alle auf dem Rechner befindlichen Datenbanken ermittelt werden.

Programmtext

DatenbankLokation (Agent) : Aktion

Starten | Client | Java

- JavaAgent.java

```
import lotus.domino.*;

public class JavaAgent extends AgentBase {

    public void NotesMain() {
  try {
    Session session = getSession();
    AgentContext agentContext = session.getAgentContext();


// (Hier beginnt der eigentliche Code)
 ❶
    DbDirectory dir = session.getDbDirectory(null);
    String server = dir.getName();

// Fixer Text wo sich die Notes-Datenbanken befinden - wenn kein Server vorhanden ist
// wird "lokal" als Speicherort angegeben
 ❷
   if (server.equals("")) server = "Lokal";
    System.out.println ("Notes Datenbanken auf dem Server "
      + server + "\n");

 // Ermitteln der ersten Datenbank auf dem Rechner - hier nur Datenbanken!
  ❸
    Database db = dir.getFirstDatabase(DbDirectory.DATABASE);

// Durchlaufen einer Schleife für alle Datenbank auf dem Rechner

    while (db != null) {
      String fn = db.getFileName();
      String title = db.getTitle();
      System.out.println(fn.toUpperCase() + " - " + title);
      db = dir.getNextDatabase(); }  ❹

  } catch(Exception e) {
    e.printStackTrace();
  }
 }
}
```

Listung Datenbanken

Schauen wir uns dieses Code-Beispiel näher an:

1. Mit der Anweisung „*DbDirectory*“ verweisen wir auf die gleichnamige Klasse, die alle Angaben für die Ermittlung der Notes-Datenbankinformationen enthält.

2. In der ersten „if-Schleife“ wird geprüft, ob auf dem Rechner ein Domino-Server installiert ist. Ist dies nicht der Fall, so wird als Speicherlokation „Lokal“ ausgegeben.

3. Mit der Funktion „*getFirstDatabase*“ wird die erste Datenbank auf dem Rechner ermittelt. Befindet sich auf dem Rechner keine Notes-Datenbank, so wird als Ergebnis der Funktion der Wert „null“ zurückgegeben. Existiert mindestens eine Notes-Datenbank, dann wird eine Instanz der Klasse „*Database*“ erzeugt. Durch die Angabe der Konstanten (oder des Parameters) „*DATABASE*“ wird angegeben, dass wir nur nach Datenbanken und nicht etwa nach Schablonen (Konstante „*TEMPLATE*“ suchen).

4. Für die Ermittlung aller weiterer Datenbanken verwenden wir die Funktion „*getNextDatabase*“. Diese Funktion besitzt im Gegensatz zur der Funktion „g*etFirstDatabase*“ keine Parameter und verarbeitet alle weiteren Datenbanken auf dem Rechner bis der Rückgabewert der Datenbankermittlung “Null” ist.

Mit der Funktion „*getFirstDatabase*“ können unterschiedliche Parameter mitgegeben werden:

- DATABASE
  - ermittelt alle Datenbanken (.nsf)

6.5.3.1 Notes-Datenbank öffnen

In dem Beispiel des vorhergehenden Abschnitts haben wir uns angesehen, wie wir eine Liste aller auf einem Rechner installierten Domino/Notes-Datenbanken erhalten. Natürlich ist die Ermittlung des oder der Dateinamen nur eine Vorstufe zur eigentlichen Arbeit mit einer Notes-Datenbank. Denn die Information, wo die Datenbank gespeichert ist und wie sie heißt – oder auch die Information der Replikations-ID der Datenbank – wird für die eigentliche Arbeit mit einer Datenbank benötigt.

Um mit den Informationen aus einer Notes-Datenbank arbeiten zu können, muss die Datenbank zuerst geöffnet werden. Java bietet dazu einige unterschiedliche Möglichkeiten.

Eine dieser Möglichkeiten des Öffnens einer Notes-Datenbank besteht in der Verwendung der Methode „*Open*“ aus der Klasse „*Database*“. Mit dieser Methode werden die Ergebnisse der Methoden „*getFirstDatabase*“ und „*getNextDatabase*“ – dabei handelt es sich genau genommen um Referenzen auf eine Datenbank – weiter genutzt, indem diese für die weitere Verarbeitung geöffnet werden.

In dem nachfolgenden Beispiel haben wir unseren Code, den wir zur Ermittlung der Notes-Datenbanken auf einem Rechner erstellt haben, um die entsprechenden Code-Zeilen für das Öffnen der Datenbanken erweitert.

Dazu wurde lediglich die Zeile „*db.open ( )*;“ eingefügt, mit der das Öffnen einer Datenbank initiiert wird.

Zur Orientierung geben wir für eine Anzeige (für unseren Test in der Java-Debugger-Konsole) noch einen Text ein, der das Öffnen der Datenbank anzeigt. Dazu fügen wir die Zeile

```
System.out.println("Datenbank  " + " " + title + " wurde geöffnet");
```

ein.

Der komplette Code zum Öffnen aller Datenbanken auf dem Rechner sieht wie folgt aus:

Programmtext

Datenbank öffnen (Agent) : Aktion

Starten | Client | Java

- JavaAgent.java

```
import lotus.domino.*;


public class JavaAgent extends AgentBase {

    public void NotesMain() {
  try {
    Session session = getSession();
    AgentContext agentContext = session.getAgentContext();


// (Hier beginnt der eigentliche Code)

    DbDirectory dir = session.getDbDirectory(null);
    String server = dir.getName();

// Fixer Text wo sich die Notes-Datenbanken befinden - wenn kein Server vorhanden ist
// wird "lokal" als Speicherort angegeben

   if (server.equals("")) server = "Lokal";
    System.out.println ("Notes Datenbanken auf dem Server "
      + server + "\n");

 // Ermitteln der ersten Datenbank auf dem Rechner - hier nur Datenbanken!

    Database db = dir.getFirstDatabase(DbDirectory.DATABASE);

 // Durchlaufen einer Schleife für alle Datenbank auf dem Rechner

    while (db != null) {
     String fn = db.getFileName();
     String title = db.getTitle();
     System.out.println(fn.toUpperCase() + " - " + title);

   db.open( );
         System.out.println("Datenbank  " + " " + title + " wurde geöffnet");

    db = dir.getNextDatabase(); }

  } catch(Exception e) {
   e.printStackTrace();
  }
 }
}
```

Codebeispiel Datenbanken öffnen

Anhand dieses kleinen Beispiels haben wir nun gesehen, wie einfach es sein kann, Datenbanken zu öffnen – nur ist dieses Vorgehen in der Praxis wenig geeignet, um eine einzelne oder zumindest einige wenige Datenbanken für eine Verarbeitung der darin enthaltenen Informationen zu öffnen. Dieses Beispiel sollte auch lediglich dazu dienen, den Umgang mit den Klassen und Methoden kurz kennenzulernen.

Im nächsten Beispiel widmen wir uns dem wohl gängigsten Verfahren beim Öffnen von Notes-Datenbanken – nämlich dem Öffnen einer bestimmten Datenbank auf einem Rechner. Für das Öffnen einer Datenbank benötigen wir die folgenden Informationen:

- Name der Datenbank
- Speicherort der Datenbank (eventuell den Servernamen)
- optional die Replikations-ID der Datenbank

6.5.3.1.1 Datenbank mit Datenbankname öffnen

Notes-Datenbanken können gezielt durch die Angabe des Datenbanknamens geöffnet werden. Bei der Java-Programmierung kann dabei die Methode „*getDatabase*“ verwendet werden, die im Gegensatz zu den in den vorhergehenden Abschnitten vorgestellten Methoden „*getFirstDatabase*“ und „*getNextDatabase*“ einige Parameter für die korrekte Ausführung der Methode benötigt.

Die Methode „*getDatabase*“ erwartet als Parameter:

- den Datenbanknamen
- den Servernamen, auf dem sich die Datenbank befindet
- Angabe, ob die Datei bei Nicht-Vorhandensein erstellt werden soll
  - dieser Parameter kann optional verwendet werden!

Im nachfolgenden Beispiel öffnen wir die Datenbank „names.nsf“ auf dem lokalen Rechner. Für das Öffnen dieser Datenbank verwenden wir die Methode „*getDatabase*“, der wir durch die Angabe der Parameter ("", "*names.nsf*") mitteilen, dass

- der lokale Rechner verwendet werden soll – dafür stehen die beiden Hochkommata ""
- die Datenbank „names.nsf“ verwendet werden soll.

Programmtext

bestimmte Datenbank öffnen (Agent) : Aktion

Starten Client Java

- JavaAgent.java

```
import lotus.domino.*;

public class JavaAgent extends AgentBase {

 public void NotesMain() {

  try {
   Session session = getSession();
   AgentContext agentContext = session.getAgentContext();

   // Hier beginnt der Code für dieses Beispiel

   Database db = session.getDatabase(" ", "names.nsf");

   if (!db.isOpen())
    System.out.println("Die Datenbank names.nsf existiert nicht auf diesem Server");
   else
    System.out.println("Names.nsf: \"" + db.getTitle()+ "\"") ;

} catch(Exception e) {
   e.printStackTrace();


  }
 }
}
```

Spezielle Datenbank öffnen

Sollte die angegebene Datenbank wider Erwarten nicht auf dem Server vorhanden sein, so wird eine entsprechende Fehlermeldung ausgegeben, die auf das Fehlen der Datenbank hinweist. Wie bereits beschrieben, kann bei der Methode „getDatabase" der optionale Parameter „Datenbank bei Nichtvorhandensein erstellen" angegeben werden.

Wir wollen in unserem Beispiel davon keinen Gebrauch machen.

Eine weitere Möglichkeit zum Öffnen einer Datenbank unter Angabe des Datenbanknamens bietet die Methode „*openDatabase*", die sich in der Klasse „*DbDirectory*" befindet. Diese Methode kann alternativ zu der zuvor vorgestellten Methode „*getDatabase*" verwendet werden.

Im Gegensatz zu der Methode „*getDatabase*" der Klasse „Session" geben wir bei der Methode „*openDatabase*" nur den Namen der Datenbank und optional einen Parameter mit, der beim Nichtvorhandensein der angegebenen Datenbank diese erstellt.

Programmtext

OpenDatabase (Agent) : Aktion

Starten: Client | Java

- JavaAgent.java

```
import lotus.domino.*;

public class JavaAgent extends AgentBase {

  public void NotesMain() {

    try {
      Session session = getSession();
      AgentContext agentContext = session.getAgentContext();


      // Hier beginnt der eigentliche Code für openDatabase

     DbDirectory dir = session.getDbDirectory(null);

      Database db = dir.openDatabase("names.nsf");

      System.out.println(db.getTitle());


    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

OpenDatabase

Wir sehen in diesem Code-Beispiel, dass wir die Klasse „*DbDirectory*“ verwenden und anschließend deren Methode „*openDatabase*“ zum Öffnen der Datenbank „names.nsf“ nutzen.

Fassen wir noch einmal zusammen:

Für das Öffnen einer Datenbank unter Verwendung des Datenbanknamens können wir zwei unterschiedliche Standard-Methoden in Java nutzen:

- Klasse „Session2
  - Methode „getDatabase“
- Klasse „DbDirectory“
  - Methode „openDatabase“

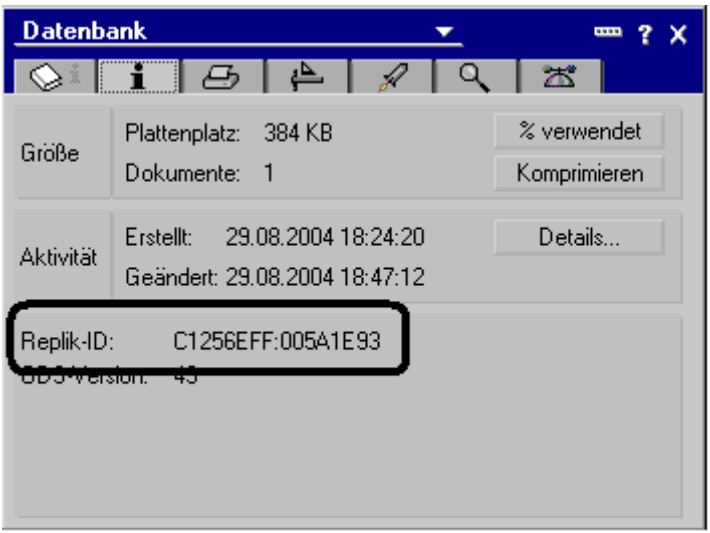
Es ist sicher Gewohnheitssache, welche dieser beiden Möglichkeiten genutzt wird.

Unter bestimmten Bedingungen kann die Angabe des Datenbanknamens gewisse Probleme mit sich bringen – dies ist dann der Fall, wenn aufgrund von Mehrfachvorkommen einer Datenbank die genaue Identität dieser Datenbank durch die Angabe des Datenbanknamens nicht in jedem Fall sichergestellt werden kann. In solchen Fällen macht die Verwendung der Replikations-ID der Datenbank Sinn, die für jede einzelne Datenbank eindeutig ist.

Um die Replikations-ID einer Datenbank zu ermitteln, können wir entweder Java-Methoden verwenden oder einfach in den Eigenschaften der Datenbank nachschauen:

❶❷❸ Ausführungsschritte:

- Öffnen Sie die entsprechende Datenbank.
- Wählen Sie in der oberen Leiste „Datei / Datenbank / Eigenschaften“ aus.
- Wechseln Sie in dem sich öffnenden Fenster in den Bereich „Info“.
  – Hier finden Sie u. a. die Replikations-ID der Datenbank.



Replik-ID

Achten Sie unbedingt darauf, dass Sie bei der Verwendung der Replik-ID nicht etwa das Trennzeichen in Form des Doppelpunktes mit in den Java-Code eingeben – denn dann würde der Zugriff auf die Datenbank fehlschlagen!

6.5.3.1.2 Datenbank mit Replik-ID öffnen

Das Öffnen einer Notes-Datenbank unter Verwendung der Replikations-ID der jeweiligen Datenbank erfolgt ähnlich wie das Öffnen einer Notes-Datenbank unter Angabe des Datenbanknamens.

Grundlage für das Öffnen mit Hilfe der Replik-ID ist die Methode „*openDatabaseByReplicaID*“, die sich in der Klasse „*DbDirectory*“ befindet.

Dieser Methode muss als Parameter die Replik-ID der Datenbank mitgegeben werden, die geöffnet werden soll. Das nachfolgende Code-Beispiel zeigt, wie eine Datenbank mit dieser Methode geöffnet werden kann.

Programmtext

OpenDatabaseByReplicaID (Agent) : Aktion

Starten | Client | Java

- JavaAgent.java

```
import lotus.domino.*;
public class JavaAgent extends AgentBase {

public void NotesMain() {

  try {
    Session session = getSession();

    AgentContext agentContext = session.getAgentContext();


    // Hier beginnt der Code, mit dem eine Datenbank mit der Replik ID
    // geöffnet wird

    DbDirectory dir = session.getDbDirectory(null);

    String rid = "C1256EFF005A1E93";

Database db = dir.openDatabaseByReplicaID(rid);
    if (db != null)

      System.out.println("Datenbank wurde geöffnet:  " + db.getTitle( ));
    else
      System.out.println("Datenbank nicht vorhanden!");


  } catch(Exception e) {
    e.printStackTrace();
  }
 }
}
```

Verwenden der Replik-ID

Das Ergebnis dieses Code-Beispiels liefert eine Referenz auf die geöffnete Datenbank.

Es kommt sicher immer auf den individuellen Fall an, ob man zum Öffnen einer Datenbank

- den Datenbanknamen und den Speicherort

oder

- die Replikations-ID der Datenbank

angeben möchte. In beiden Fällen ist das Ergebnis dasselbe: es wird eine Referenz auf eine geöffnete Datenbank erstellt.

Neben diesen allgemein gültigen Möglichkeiten zum Öffnen einer Datenbank gibt es noch einige spezielle Methoden und Klassen, die auf die Standard-Notes-Datenbanken ausgelegt sind. Dies sind u. a.:

- getAddressBooks
  - für die Ermittlung der Adressbücher auf einem Rechner
- openMailDatabase
  - für das Öffnen einer Mail-Datenbank

Diese werden wir uns in einem späteren Abschnitt näher ansehen.

Diese Seite ist zur Zeit nicht belegt.

6.5.4 Dokumente und Java

Wie bereits zu Anfang dieses Kapitels beschrieben, wollen wir versuchen, die Verwendung von Java im Notes/Domino-Umfeld mit Hilfe der einzelnen Notes-Bestandteile zu beschreiben. Im vorhergehenden Abschnitt haben wir uns angesehen, wie die Behandlung von Notes-Datenbanken mit Hilfe von Java erfolgen kann.

Als nächstes wollen wir uns die Verarbeitung von Dokumenten innerhalb dieser Datenbanken ansehen.

Ähnlich wie für den Zugriff und die Verarbeitung der Datenbanken mit Hilfe der Klasse „*Database*“ existiert für die Verarbeitung der Notes-Dokumente eine eigene Klasse „*Document*“. In dieser Klasse befinden sich alle Methoden und Eigenschaften für den Umgang mit Dokumenten innerhalb von Java-Anwendungen.

Bei der Verarbeitung von Dokumenten müssen wir zunächst die Datenbank öffnen, in der sich das Dokument (oder die Dokumente) befinden. Den Zugriff auf Datenbanken haben wir uns im vorhergehenden Abschnitt angesehen. Auf dieses Wissen bauen wir in diesem Abschnitt auf.

Um auf Dokumente zugreifen zu können, benötigen wir folgende Informationen:

- In welcher Datenbank befindet sich das Dokument?
- Wo ist die Datenbank gespeichert?
- Wie kann diese Datenbank geöffnet werden, damit auf deren Inhalt zugegriffen werden kann?

- Wie kann das Dokument gefunden werden?
- Wie kann auf das Dokument zugegriffen werden?

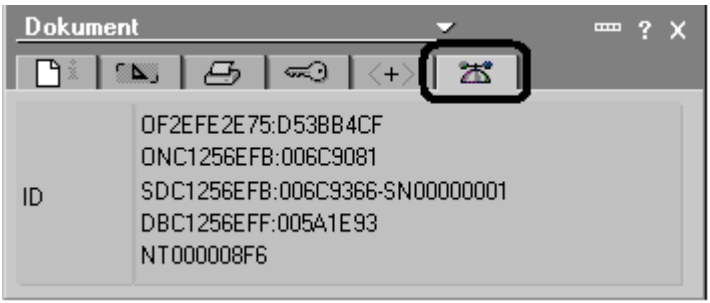
Die ersten drei Fragen können wir bereits mit Hilfe der Informationen aus dem Vorkapitel beantworten. Also beschäftigen wir uns mit der Fragestellung:

„Wie können wir ein Dokument in einer Datenbank finden?“

Machen wir uns zunächst klar, wie ein Dokument innerhalb einer Datenbank gespeichert und eindeutig identifiziert wird:

Ähnlich wie eine Datenbank enthält jedes Dokument ein eindeutiges Kennzeichen, das für die Identifikation jedes Dokumentes verwendet werden kann – die Dokument-ID

Die Dokument-ID finden wir unter den Eigenschaften für das jeweilige Dokument. Diese können wir uns in Notes zum Beispiel über einen Klick mit der rechten Maustaste in einem geöffneten Dokument ansehen:



Dokument-ID

Sie sehen, die Dokument-ID ist eine recht komplexe Ziffern-/ Zahlenfolge.

6.5.4.1 Ein Dokument mit Dokument-ID finden

Damit wir nun ein Dokument mit der dazugehörigen ID finden können, verwenden wir die Methode „*getDocumentByID*“, die sich in der Klasse „*Database*“ befindet. Wie bereits beim Verarbeiten der Datenbanken, so ist das Ergebnis der Ausführung einer solchen Methode eine Instanz der Klasse – in diesem Fall der Klasse „*Document*“.

Schauen wir uns zunächst einmal die ID eines Dokuments in der Abbildung „Dokument-ID“ an. Die gesamte Folge aller Ziffern und Buchstaben stellt diese ID dar.

Nun wäre die Angabe dieser Zeichenfolge zur Identifikation eines Dokumentes wohl mehr als unpraktikabel. Damit eine Dokument-ID nicht manuell eingegeben werden muss, kann diese auch durch einen entsprechenden Zugriff auf ein Dokument ermittelt werden. Dazu bietet Lotus die Methode „*getNodeID*“ an. In dem folgenden Beispiel wird aus einem geöffneten Dokument heraus der Agent aufgerufen.

Sie erkennen in diesem Code-Beispiel bereits einige wesentliche Elemente für das Arbeiten mit Dokumenten innerhalb von Java: Wir wollen an dieser Stelle noch nicht auf alle Elemente dieses Beispiel-Programms eingehen. Wir schauen uns das Statement an, das die Notes-ID des Dokuments ermittelt. Diese Stelle ist im Beispiel entsprechend markiert.

Programmtext

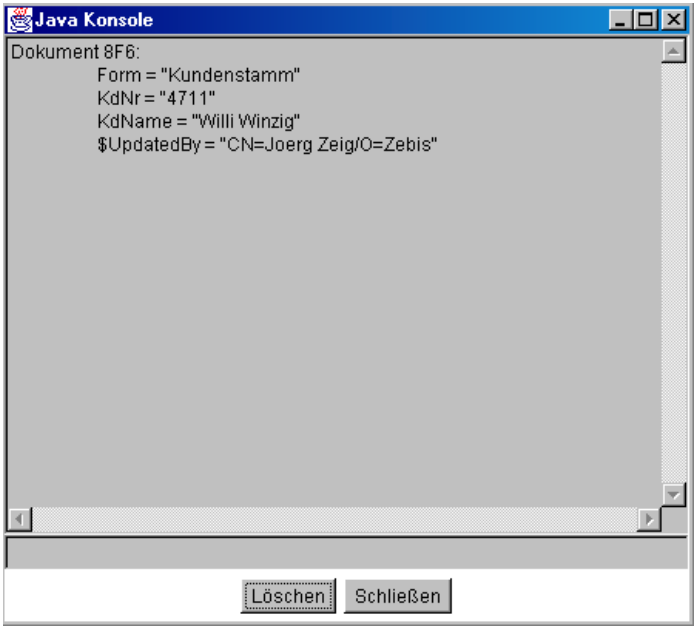
```
import lotus.domino.*;
import java.util.Vector;
public class JavaAgent extends AgentBase {
 public void NotesMain() {

  try {

   Session session = getSession();
   AgentContext agentContext = session.getAgentContext();

  // Hier geht´s los
   Database db = agentContext.getCurrentDatabase();
   DocumentCollection dc = db.getAllDocuments();
   Document doc = dc.getFirstDocument();
   while (doc != null) {

System.out.println("Dokument " +
  doc.getNoteID() + ":");
 Vector items = doc.getItems();
    for (int j=0; j<items.size(); j++) {
     Item item = (Item)items.elementAt(j);
     System.out.println("\t" +
     item.getName() + " = \"" +
         item.getValueString() + "\"");
     }

    doc = dc.getNextDocument();
    }
  } catch(Exception e) {
   e.printStackTrace();
  }
 }
}
```

getNoteID

Durch die Angabe „*doc.getNoteID*“ wird das Programm veranlasst, die ID des gerade in Zugriff befindlichen Dokuments als Ergebnis zu liefern. Diese ID kann dann im weiteren Verlauf des Programms verwendet werden.

Zur Verdeutlichung dieses Agenten lassen wir das Ergebnis in der Java-Debug-Konsole anzeigen. Hier geben wir die über die Dokument-ID ermittelten Werte

- Maske
- Kunden-Nummer
- Kunden-Name
- geändert von



Dokument-Informationen

6.5.4.2 Ein Dokument mit Dokument-UNID finden

Jedes Notes-Dokument enthält neben der gerade beschriebenen Dokument-ID noch eine weitere ID – die „*Universal-ID*“. Diese ID kann ähnlich wie die Dokument-ID für den Zugriff mittels Java verwendet werden. Die entsprechenden Werkzeuge dazu finden wir in der Methode „*getDocumentByUNID*“, die sich ebenfalls in der Klasse „*Database*“ befindet.

Als Parameter muss dieser Methode die Dokument-UNID mitgegeben werden. Das Ergebnis einer erfolgreichen Ausführung der Methode „*getDocumentByUNID*“ ist dann eine Instanz des gefundenen Dokuments.

Achten Sie bei der Verwendung der Dokument-UNID darauf, dass diese eine fixe Länge von 32 Zeichen hat!

Der Aufruf dieser Methode kann wie folgt aussehen:

```
Document doc = db.getDocumentByUNID (id)
```

Es gibt noch einige weitere Methoden, die sich auf die Dokument-UNID beziehen. Die wichtigste dürfte die Methode „*getUniversalID*“ sein, mit der die Dokument-UNID eines Dokuments ermittelt werden kann. Diese Methode befindet sich in der Klasse „*Docu-*

ment“ – (nicht in der Klasse „*Database*“ wie z. B. die Methode „*getDocumentByUNID*“!

Programmtext

```
- JavaAgent.java
import lotus.domino.*;

public class JavaAgent extends AgentBase {

 public void NotesMain() {

  try {
   Session session = getSession();
   AgentContext agentContext = session.getAgentContext();

   // (Hier beginnt der Code

   DocumentCollection dc = db.getAllDocuments();
   Document doc = dc.getFirstDocument();

   while (doc != null) {                    ①

   String id = "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH";

      Document pdoc =

   db.getDocumentByUNID(id);                ②

      String docSubj = doc.getItemValueString("KdNr");
      String pdocSubj = pdoc.getItemValueString("KdName");

   System.out.println ("Dokument gefunden" + pdocSubj);

 doc = dc.getNextDocument(doc);
   }

  } catch(Exception e) {
   e.printStackTrace();
  }
 }
}
```

Dokument-UNID verwenden

In dem Programmbeispiel oben suchen wir ein Dokument anhand einer bekannten Dokument-ID – in dem Beispiel oben ist dies die Dok-UNID

„*AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH*“

Dieser ordnen wir die Variable „id“ zu (Punkt ①), um mit dieser dann in einem weiteren Schritt ② mit der Methode „getDocumentByUNID“ das Dokument zu ermitteln, das diese UNID enthält.

Diese Möglichkeiten haben nun beschrieben, wie auf ein bestimmtes Dokument innerhalb einer Datenbank zugegriffen werden kann. In der Praxis ist es oft so, dass entweder alle Dokumente einer Datenbank oder zumindest einige bestimmte Dokumente einer Datenbank verarbeitet werden müssen, die bestimmten Kriterien entsprechen.

Die Zugriffsmöglichkeiten auf diese Dokumente werden wir uns nun näher ansehen.

6.5.4.3 Dokumentensammlungen verwenden

Neben dem direkten Zugriff auf ein bestimmtes Dokument innerhalb einer Notes-Datenbank werden in der Praxis meist Selektionen von mehreren Dokumenten benötigt. Bei solchen Dokumenten spricht man auch von einer „*Dokumentensammlung*“ oder „*Document Collection*“.

Für die Verarbeitung von Dokumentensammlungen bietet Lotus die Klasse „*DocumentCollection*“, die verschiedene Methoden zur Verarbeitung von mehreren Dokumenten enthält.

Beginnen wir mit der ersten Methode, die für den Zugriff auf alle Dokumente erforderlich ist – der Methode „*getFirstDocument*“. Diese Methode wird ohne jegliche Parameter aufgerufen und liest das erste Dokument einer Dokumentensammlung ein.

Wir kennen es bereits von der Verwendung der bisher gezeigten Methoden: Der Rückgabewert der Methode ist eine Instanz der Klasse „*Document*“ – quasi eine Referenzierung auf das erste Dokument in der Dokumentensammlung.

Schauen wir uns dies einmal an dem nächsten Programmbeispiel an:

Programmtext

- JavaAgent.java

```
import lotus.domino.*;
public class JavaAgent extends AgentBase {
 public void NotesMain() {
  try {
    Session session = getSession();
    AgentContext agentContext =
    session.getAgentContext();

// Hier beginnt der Code

    Database db = agentContext.getCurrentDatabase();
    DocumentCollection dc = db.getAllDocuments();

    Document doc = dc.getFirstDocument();        ①

while (doc != null) {

     System.out.println(doc.getItemValueString
     ("KdName"));

doc = dc.getNextDocument(); }        ②

  } catch(Exception e) {
    e.printStackTrace();
  }
 }
}
```

getFirstDocument

In diesem Beispiel ermitteln wir mit der Methode „*getFirstDocument*“ das erste Dokument in einer Dokumentensammlung, die wir durch die Angabe „*db.getAllDocuments*“ erstellt haben.

Nachdem das erste Dokument ermittelt wurde, wird der Inhalt des Feldes „*KdName*“ ausgegeben. Danach wird dann mit einer weiteren Methode der Klasse „DocumentCollection“ – der Methode „getNextDocument“ (Schritt k in der Abbildung) – das nächste Dokument der Sammlung eingelesen. Durch die Schleifenprogrammierung wird auch der Feldinhalt „*KdName*“ aller mit dieser Methode eingelesenen Dokumente ausgegeben.

Die Methode „getNextDocument“ liest, wie wir eben gesehen haben, das nächste Dokument einer Dokumentensammlung ein. Im Gegensatz zu der Methode „getFirstDocument“ kann hier ein Parameter mitgegeben werden, der steuert, von welchem Dokument das Folgedokument ermittelt werden soll.

In der Praxis ist die Verwendung dieses Parameters häufig die Ursache für massive Performance-Probleme – deshalb wird vor der Verwendung des Parameters in dieser Klasse ausdrücklich gewarnt!!

Das Ergebnis der Ausführung dieses Agenten könnte so aussehen:

Ergebnis Dokumentensammlung

Das Ergebnis dieser beiden Methoden

- getFirstDocument
- getNextDocument

sind uneingeschränkt alle Dokumente der Dokumentensammlung.

Gerade bei großen Datenbanken kann die Verwendung dieser beiden Methoden durchaus Nachteile haben. Deshalb gilt es zu überlegen, welche der Dokumente verarbeitet werden müssen – alle oder einzelne Dokumente?

Im letzteren Fall werden nicht alle Dokumente einer Dokumentensammlung verarbeitet, sondern es wird mit der Methode „*getNthDocument*“ direkt auf das zu verarbeitende Dokument zugegriffen.

Um auf ein bestimmtes Dokument in einer Dokumentensammlung zugreifen zu können, muss die Position des Dokuments bekannt sein.

Im nächsten Programmbeispiel wollen wir nur ein Dokument auswählen.

Dazu geben wir zunächst einmal die Verwendung der Dokument-Sammlung an (Punkt ① in der Abbildung). Wir wissen, dass es sich bei dem Dokument um den 4. Eintrag in der Datenbank handelt – deswegen geben wir dies als Parameter für die Ausführung des Agenten mit (Punkt ② in der Abbildung). Als Rückgabe erhalten wir den Kundennamen bei der Ausführung der Methode „*getNthDocument*“ (Punkt ③ in der Abb.).

Programmtext

- JavaAgent.java

```
import lotus.domino.*;
public class JavaAgent extends AgentBase {
 public void NotesMain() {
  try {
   Session session = getSession();
   AgentContext agentContext = session.getAgentContext();
   // Hier beginnt der Code

   Database db = agentContext.getCurrentDatabase();

   DocumentCollection dc = db.getAllDocuments();      ①

  int N = 4;      ②

printDocument(dc, N);
  } catch(Exception e) {
   e.printStackTrace();
  }
 }
 void printDocument(DocumentCollection dc, int n) {
  try {
   if (n < 0 || n > dc.getCount())

    System.out.println("Positionierung fehlerhaft - keinen Datensatz gefunden");

   else

    System.out.println("Dokument " + n + ": " + "enthält Kundennamen:  " +

   dc.getNthDocument(n).getItemValueString("KdName"));      ③

  } catch(Exception e) {
   e.printStackTrace();
  }
 }
}
```

getNthDocument

6.6 XML

Datenaustausch zwischen unterschiedlichen Systemen – in diesem Zusammenhang fällt immer wieder der Begriff: „XML“.

Es gibt unterschiedliche Möglichkeiten, Domino-Daten mit Nicht-Domino-Systemen auszutauschen. Neben DECS, LEI & Co. gibt es seit dem Release 6.5 XML ein zusätzliches Werkzeug, um Domino-Daten zu exportieren oder auch zu importieren.

XML – „Extensible Markup Language“ hat sich in den letzten Jahren zu einem Standard für Datenaustauschprozeduren entwickelt. Dabei stellt XML eine Art Datenbeschreibung dar, die die einzelnen Elemente und Inhalte umschreibt und für andere Systeme verfügbar macht.

Die Integration und die Nutzung von XML in Domino-/Notes-Anwendungen waren bis zum Erscheinen von R6.5 noch recht rudimentär. Bereits mit Release 5 wurden die ersten Ansätze der XML-Integration verfügbar. Mit R6.5 wurde die Integration in Form von eigenen Klassen in LotusScript und JavaScript wesentlich erweitert – genau genommen wurde sie erstmalig wirklich praktikabel realisiert.

Zwar ist XML ein Standard, aber die Integration in das jeweilige System erfolgt meist individuell – so auch die XML-Integration in Domino. Diese wird realisiert durch „DXL“.

DXL ist eine spezielle Form der Datenbeschreibung für Notes-spezifische Daten, Notes-Elemente (Masken, Ansichten etc.) und deren Inhalte.

Standardisierte Import- und Export-Funktionen ermöglichen den Transfer der Domino-spezifischen XML-Angaben in andere XML-Formate und bilden damit die Grundlage für den Datenaustausch zwischen Domino und Nicht-Domino-Anwendungen.

Im Zusammenhang mit XML fallen einige Fachbegriffe, die nachfolgend kurz erläutert werden sollen. Ohne sie ist ein Verständnis der XML-Arbeitsweise nur schwer möglich.

DTD

DTD steht für **D**ocument **T**ype **D**efinition und enthält die Beschreibung der Datenstrukturen von XML. DTD ist ein von Domino unabhängiger Begriff und allgemeingültig in der XML-Welt.

Unter anderem werden in einer DTD definiert:

- **Entities**
  Unter anderem sind dies Definitionen von Datentypen.
- **Elemente**
  Beschreiben, welche Tags in einem XML-Dokument verwendet werden können und welche Struktur diese haben.

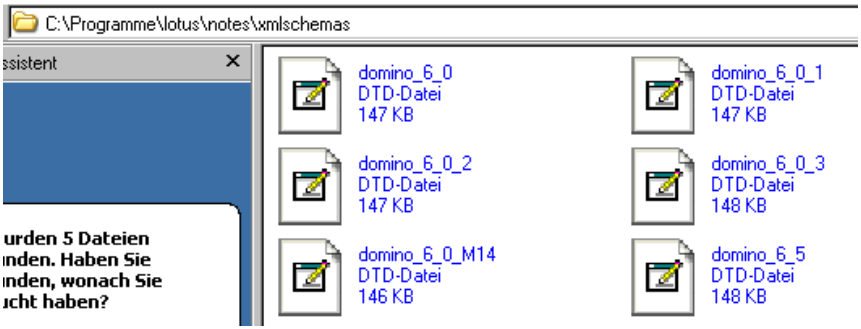
DTD stellt die XML-Struktur in einer Baumstruktur dar.

Für Domino gibt es eine spezielle DTD, welche die Domino-spezifischen Informationen enthält – die Domino DTD. Das mit dieser DTD erzeugte Ergebnis wird dann als DXL (Domino Extented Language) bezeichnet.

Betrachtet man die Datenstruktur von Domino, dann ist diese im Vergleich zu anderen Systemen sehr komplex – zwangsläufig ist auch die DTD für Domino sehr komplex.

Die Domino DTD wurde bereits in R5 eingeführt – für die Optimierung in R6 allerdings nochmals geändert. Für jedes Domino-Release kann es eine eigene DTD geben (siehe Abb. „DTD Versionen“).

Im Basisverzeichnis von Notes sind die einzelnen DTD-Versionen installiert.



DTD-Versionen

Das nachfolgende Beispiel zeigt einen kleinen Ausschnitt der Domino DTD.

```
00001<!--
00002================================================================
00003                        DOMINO_6_5.DTD
00004
00005This DTD describes the structure of DXL, the Domino XML Language,
00006for Notes/Domino version 6.5
00007
00008Copyright (c) 1999-2003 IBM.
00009All rights reserved.
00010
00011NOTE:
00012
00013Some miscellaneous types of data that can appear only in design-
00014element notes of a Domino database are not yet handled by the
00015current versions of the DXL exporter and importer programs (for
00016example, custom datetime information in view notes).  If a specific
00017type of Domino data is defined in this DTD, then the DXL exporter
00018and importer programs can handle it and, conversely, if a specific
00019type of Domino data is not defined in this DTD, then those programs
00020cannot handle it yet.
00021
00022================================================================
00023-->
00024
00025<!-- ***************************************************** -->
00026<!-- * Common                                              * -->
00027<!-- ***************************************************** -->
00028
00029<!-- **** Elements **** -->
00030
00031<!ELEMENT created ( datetime )>
00032
00033<!ELEMENT modified ( datetime )>
00034
00035<!-- **** Types **** -->
00036
00037<!ENTITY % keywords.ui "
00038    dialoglist |
00039    checkbox |
00040    radiobutton |
00041    combobox |
00042    listbox
00043    ">
00044
00045<!ENTITY % keywords.borders "
00046    none |
00047    single |
00048    inset
00049    ">
00050
00051<!-- ***************************************************** -->
00052<!-- * Basic Types                                         * -->
00053<!-- ***************************************************** -->
00054
00055<!ENTITY % boolean.choices "
00056    true |
00057    false
00058    ">
```

DTD-Beispiel

DXL

DXL oder auch „Domino Extented Language“ ist das Ergebnis der mit der Domino-DTD erstellten Versionen und stellt die Basis der XML-Integration in Domino dar. Dabei werden u. a. die Elemente von Domino/Notes in Form von Masken, Ansichten etc. beschrieben und können diese z. B. in Nicht-Domino XML-Versionen umgesetzt werden.

XML Parser

XML-Daten sind streng genommen zunächst reine Textinformationen. Die Verwendung dieser Daten in Applikationen bedingt eine Interpretation der Informationen in der Art, dass diese Textinformationen als „Daten“ erkannt werden. Für diese Erkennung wird der so genannte Parser verwendet. Neben dem Aufbereiten der Informationen übernimmt der Parser auch bestimmte Prüffunktionen, über die der Inhalt der Dokumente analysiert wird. In der Literatur findet man hin und wieder auch die Bezeichnung „XML-Prozessor“. Der XML Parser für Domino ist in Form von besonderen Klassen realisiert, die in der Folge der Import-/Export-Funktionen von XML angewendet werden.

Der Parser prüft den Inhalt eines Dokuments auf Einhalten des XML-Standards und setzt die Informationen für die weitere Verarbeitung um.

Der Parser überprüft den Inhalt einer Datei auf XML-Regeln.

Das nachfolgende kleine Beispiel zeigt eine XML-Datei, die bereits einige wesentliche XML-Elemente enthält.

```
<?xml version="1.0" encoding="UTF-8"?>
<Kunde>
<KundenNr>0000004711</KundenNr>
<Nachname>Winzig</Nachname>
<Vorname>Willi</Vorname>
<Firma>KnowHow GmbH</Firma>
<Strasse>Postfach 4567</Strasse>
<PLZ>70000</PLZ>
<Stadt>Stuttgart</Stadt>
</Kunde>
```

XML-Beispiel 02

XSL Transformer

Domino verwendet, wie zuvor bereits erwähnt, eine besondere Variante von XML. Diese Informationen können nicht 1:1 mit Nicht-Domino XML ausgetauscht werden, sondern müssen zunächst angepasst werden. Diese Anpassungen werden durch die Klasse XSL Transformer durchgeführt.

DOM

Im Zusammenhang von XML und Domino fällt oft die Abkürzung „DOM“. Diese steht für **D**ocument **O**bject **M**odell. Dabei handelt es sich um ein API, mit dem auf XML-Strukturen zugegriffen werden kann. XML beschreibt zunächst nur die Struktur von Daten. Wie diese Daten verarbeitet werden und wie auf sie zugegriffen werden kann, wird in dem „Document Object Model“ hinterlegt. Dabei ist DOM ein Objektmodell, in dem Objekte mit Eigenschaften und Methoden enthalten sind. Diese werden

in eine Art Baumstruktur umgesetzt. Der Baum besteht aus einzelnen Knoten.

Für die Verarbeitung von XML in Domino gibt es einen speziellen Parser, der die Domino-spezifischen XML-Informationen verarbeitet, den „NotesDOM Parser“.

Achtung!

Im Domino Umfeld wird „Domino Object Model“ ebenfalls als „DOM“ abgekürzt – dies hat nichts mit dem „Document Object Model“ zu tun!

Sowohl für die Import- als auch für die Exportfunktionen gilt, dass die Verarbeitungsschritte für diese Funktionen aus verschiedenen Stufen bestehen.

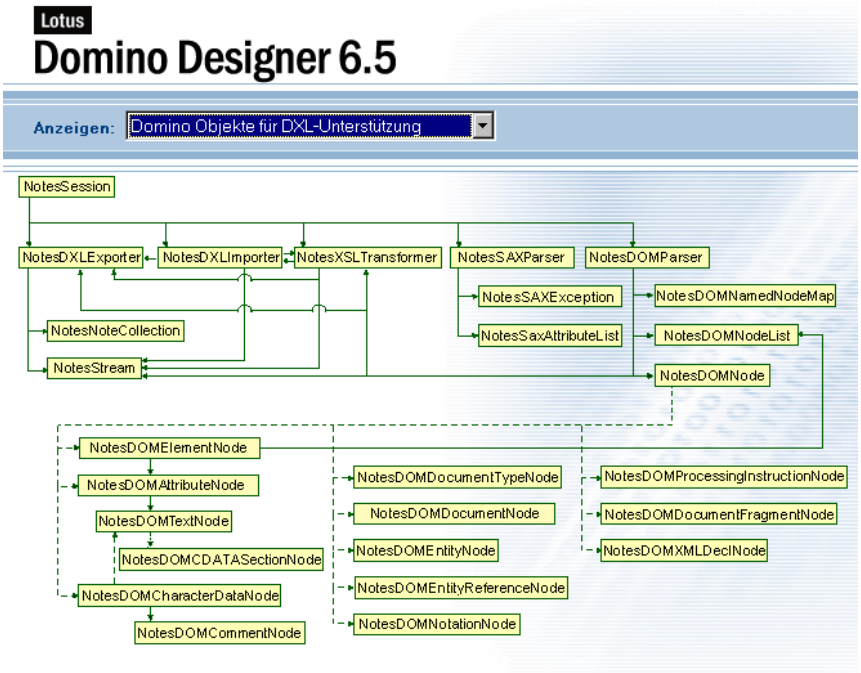
Sehen wir uns das einmal am Beispiel eines Datenexports aus Domino in XML an.

Zunächst müssen die Informationen aus der Domino-/Notes-Datenbank in das Domino-spezifische XML-Format DXL umgesetzt werden. Dazu verwenden wir zunächst die Klasse DXLExporter.

Diese Seite ist zur Zeit nicht belegt.

6.6.1 XML-Klassen

XML wird über Script- und Java-Klassen in Domino integriert. Dabei stellt die Klasse „*NotesXMLProcessor*“ die Basisklasse für die XML-Integration dar. Diese Klasse ist die Basis aller weiteren XML-Klassen und vererbt die Informationen an die abgeleiteten Klassen. Da diese Klasse nicht direkt für die XML-Einbindung verwendet werden kann, wird sie auch nicht in den Übersichten der Gestaltungselemente aufgelistet.

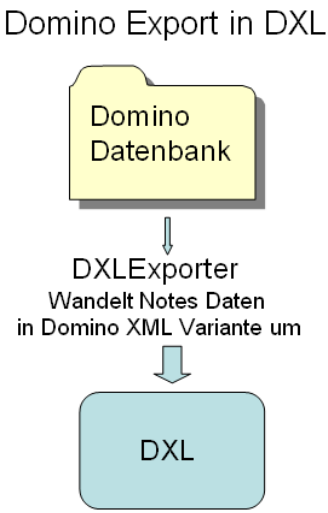


XML-Struktur

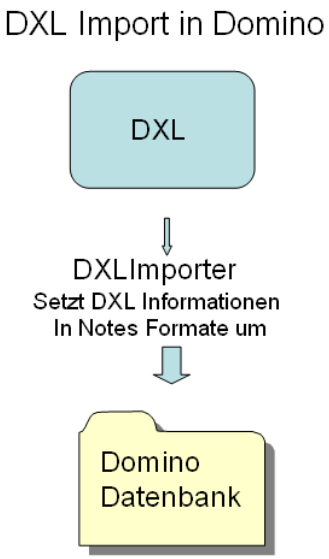
Für Import- und Export Funktionen zwischen Domino und DXL dienen die beiden Klassen:

- DXLImporter
- DXLExporter

Diese übernehmen den Informationstransfer zwischen den NSF-Dateien und den DXL-Dateien, von wo aus diese dann bei Bedarf auch in Nicht-Domino-Formate weiterverarbeitet werden können.



Export Schematisch



Import Schematisch

6.6.1.1 Export-Klasse

Der wesentliche Vorteil der Nutzung von XML in Verbindung mit Domino-Anwendungen liegt darin, Informationen mit anderen Anwendungen auszutauschen und diese möglichst ohne großen administrativen Aufwand zur Verfügung zu stellen – bzw. die Informationen anderer Systeme möglichst reibungslos in Domino zu importieren.

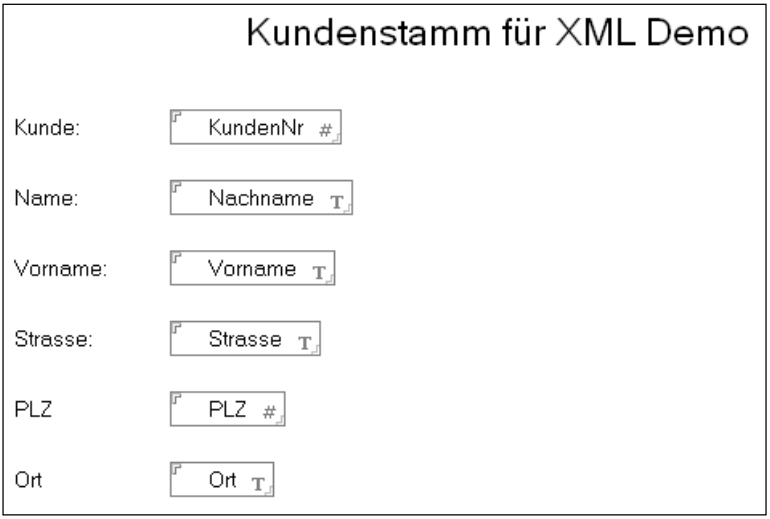
Die Basis für die Export-Funktionen findet sich in LotusScript – oder genauer in LotusScript-Klassen. Dazu wird die Script-Klasse NotesDXLExporter verwendet, die seit R6 zur Verfügung steht.

Die Klasse NotesDXLExporter kann Informationen aus den nachfolgenden Domino-Objekten verarbeiten:

- NotesDatabase
- NotesDocument
- NotesDocumentCollection
- NotesNoteCollection

Anhand eines kleinen Beispiels soll die Exportfunktion veranschaulicht werden. In diesem Beispiel werden alle Informationen einer Datenbank in eine Datei geschrieben und somit für den Export in eine Nicht-Domino-Anwendung zur Verfügung gestellt.

In diesem Beispiel wird eine Datenbank namens „XMLKunden01“ exportiert. Der Aufbau der in dieser Datenbank enthaltenen Maske sieht wie folgt aus:



Maskenaufbau

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim pfadname As String
    Set db = session.CurrentDatabase
    pfadname$ = "c:\xmldemo\"

  REM Öffnen XML Datei
    Dim stream As NotesStream
    Set stream = session.CreateStream
    filename$ = pfadname$ &  Left(db.FileName, Len(db.FileName) - 3) & "xml"
    If Not stream.Open(filename$) Then
        Messagebox "Die Datei " & filename$ & "kann nicht geöffnet werden",, "Fehler!"
        Exit Sub
    End If
    Call stream.Truncate

  REM Datenbank als XML exportieren
    Dim exporter As NotesDXLExporter
    Set exporter = session.CreateDXLExporter
    Call exporter.SetInput(db)
    Call exporter.SetOutput(stream)
    Call exporter.Process
End Sub
```

Export-Klasse

In diesem Beispiel wird die aktuelle Notes-Datenbank (XMLKunden01) in eine gleichnamige Datei exportiert und mit der Endung „.XML“ ergänzt. Als Pfad für diese Datei wird das Verzeichnis „c:/xmldemo“ verwendet. Der Pfad muss vor der Ausführung dieses Scripts bereits existieren. Wurde er nicht bereits vor dem Script-Lauf angelegt, erscheint eine entsprechende Fehlermeldung. Basis für den Export bildet die Notes-Klasse „Stream“ – die einfachste Variante, Einträge aus einer Datenbank zu verarbeiten.

Da die Export-Klasse eine Unterklasse von Notes Session ist, muss diese innerhalb einer Notes Session ausgeführt werden. Dies wird in unserem Code-Beispiel durch die erste Zeile angegeben.

Für den Export der Datenbankinformationen verwendet das Beispiel die Klasse *NotesStream*, die mit dem Befehl *CreateStream* erzeugt wird. Mit diesem Verfahren wird zuerst ein Datenstrom-Objekt erzeugt, in das die Informationen gestellt werden.

```
Dim stream As NotesStream
Set stream = session.CreateStream
```

Stream-Angaben

Mit der Anweisung

filename$ = pfadname$ & Left(db.FileName, Len(db.FileName) - 3) & „xml“

wird der Name der Exportdatei festgelegt. Dabei wird in das zuvor angegebene (und bereits erstellte!) Verzeichnis eine Datei mit dem Namen der Notes-Datenbank gestellt. Diese erhält die Endung „.XML“. Das Ergebnis dieser Anweisungszeile lautet dann: „XMLKunden01.xml“.

Das Schreiben der Informationen aus der Notes-Datei in die eben erzeugte XML-Datei erfolgt über folgende Anweisung:

stream.Open(filename$)

Um etwaige Fehler abzufangen, wird diese Anweisung in eine „IF"-Schleife eingebunden.

```
If Not stream.Open(filename$) Then
   Messagebox "Die Datei " & filename$ & "kann nicht geöffnet werden",, "Fehler!"
   Exit Sub
End If
```

Open-Angaben

Mit der Anweisung „Truncate" wird der Datenstrom zurückgesetzt.

```
Call stream.Truncate
```

Truncate

Damit sind allerdings immer noch keine Informationen in die neue Datei geschrieben worden. Alle bisherigen Angaben sind erforderliche, vorbereitende Angaben für den Export der Informationen. Den eigentlichen Export übernimmt die nächste Anweisung, in der ein Objekt der Klasse „NotesDXLExporter" erzeugt wird. Diesem wird als Information die Datenquelle – also die Notes-Datenbank (SetInput) und die Zielangabe – als die Exportdatei mitgegeben (SetOutput).

```
REM Datenbank als XML exportieren
  Dim exporter As NotesDXLExporter
  Set exporter = session.CreateDXLExporter
  Call exporter.SetInput(db)
  Call exporter.SetOutput(stream)
```

Export

Die Anweisung „Process“ startet die Exportausführung für diesen Datensatz.

```
REM Datenbank als XML exportieren
  Dim exporter As NotesDXLExporter
  Set exporter = session.CreateDXLExporter
  Call exporter.SetInput(db)
  Call exporter.SetOutput(stream)
  Call exporter.Process
End Sub
```

Process

Die Zieldatei wird nach der Ausführung des Scripts automatisch erzeugt und gespeichert.

Das Ergebnis dieses kleinen Scripts sieht dann so aus:

XML-Datei

Der Inhalt dieser Datei kann mit jedem Editor eingesehen werden.

```
00001<?xml version='1.0' encoding='utf-8'?>
00002<!DOCTYPE database SYSTEM 'xmlschemas/domino_6_5.dtd'>
00003<database xmlns='http://www.lotus.com/dxl' version='6.5' replicaid='C1256E990036A599'
00004 path='XMLKunden01.nsf' title='XMLKunden01'>
00005<databaseinfo dbid='C1256E990036A599' odsversion='43' diskspace='327680' percentused='86.640625'
00006 numberofdocuments='1'><datamodified><datetime dst='true'>20040519T115810,85+02</datetime></datamodified><de
00007><datetime dst='true'>20040519T115745,96+02</datetime></designmodified></databaseinfo>
00008<acl maxinternetaccess='editor'>
00009<aclentry name='-Default-' default='true' level='noaccess' readpublicdocs='true'
00010 writepublicdocs='true'/>
00011<aclentry name='OtherDomainServers' type='servergroup' level='noaccess' readpublicdocs='false'
00012 writepublicdocs='false'/>
00013<aclentry name='CN=Joerg Zeig/O=Zebis' type='person' level='manager' deletedocs='true'/>
00014<aclentry name='LocalDomainServers' type='servergroup' level='manager' deletedocs='true'/>
00015<logentry>19/05/2004 11:56:53 AM Joerg Zeig/Zebis f[illegible]gte OtherDomainServers hinzu</logentry>
00016<logentry>19/05/2004 11:56:53 AM Joerg Zeig/Zebis f[illegible]gte LocalDomainServers hinzu</logentry>
00017<logentry>19/05/2004 11:56:53 AM Joerg Zeig/Zebis aktualisierte Joerg Zeig/Zebis</logentry>
00018<logentry>19/05/2004 11:56:53 AM Joerg Zeig/Zebis f[illegible]gte Joerg Zeig/Zebis hinzu</logentry>
00019<logentry>19/05/2004 11:56:53 AM Joerg Zeig/Zebis aktualisierte -Default-</logentry></acl>
00020<view showinmenu='true' publicaccess='false' default='true' noviewformat='true'>
00021<noteinfo noteid='116' unid='1A787D917B4C392EC1256E990036A5B7' sequence='1'>
00022<created><datetime dst='true'>20040519T115653,67+02</datetime></created>
00023<modified><datetime dst='true'>20040519T115745,96+02</datetime></modified>
00024<revised><datetime dst='true'>20040519T115653,67+02</datetime></revised>
00025<lastaccessed><datetime dst='true'>20040519T115745,96+02</datetime></lastaccessed>
00026<addedtofile><datetime dst='true'>20040519T115653,67+02</datetime></addedtofile></noteinfo>
00027<updatedby><name>CN=Joerg Zeig/O=Zebis</name></updatedby><code event='selection'><formula
00028>@All</formula></code></view>
00029<note default='true' class='icon'>
00030<noteinfo noteid='11a' unid='A59E9BAEEA53A51CC1256E990036A5B9' sequence='3'>
00031<created><datetime dst='true'>20040519T115653,69+02</datetime></created>
00032<modified><datetime dst='true'>20040519T115653,72+02</datetime></modified>
00033<revised><datetime dst='true'>20040519T115653,71+02</datetime></revised>
00034<lastaccessed><datetime dst='true'>20040519T115653,71+02</datetime></lastaccessed>
00035<addedtofile><datetime dst='true'>20040519T115653,69+02</datetime></addedtofile></noteinfo>
00036<updatedby><name>CN=Joerg Zeig/O=Zebis</name></updatedby>
```

XML Datei-Inhalt

Dabei fällt die Ähnlichkeit mit dem Aufbau von HTML-Dateien auf, allerdings ist die Syntax nicht identisch mit der von HTML.

Schaut man sich die Einträge in der XML-Datei (oder besser der DXL-Datei, denn es handelt sich um eine Domino-spezifische XML-Datei) an, dann erkennt man u. a. auch die einzelnen Domino-spezifischen Einträge – wie zum Beispiel ACL-Einträge, Maskenbeschreibungen, Beschreibung der Felder der Maske etc.

```
00130<updatedby><name>CN=Joerg Zeig/O=Zebis</nam
00131<document form='KundenstammXMLDemo'>
[redacted]
00133<created><datetime dst='true'>20040519T1157
```

Maskeninformation

```
00138<updatedby><name>CN=Joerg Zeig/O=Zebis</name></updatedby>
00139<item name='KundenNr'><number>1</number></item>
00140<item name='Nachname'><text>Winzig</text></item>
00141<item name='Vorname'><text>Willi</text></item>
00142<item name='Strasse'><text>Hauptstrasse 47</text></item>
00143<item name='PLZ'><number>40000</number></item>
00144<item name='Ort'><text>Düsseldorf</text></item></document></database>
00145
```

Felder in XML

Neben den einzelnen Feldern und Dokumenten enthält diese globale DXL u. a. auch die Angaben der Zugriffskontroll-Liste ACL, ein sicher nicht in jedem Fall sinnvoller Effekt – doch dazu später mehr.

```
<acl maxinternetaccess='editor'>
<aclentry name='-Default-' default='true' level='noaccess' readpublicdocs='true'
 writepublicdocs='true'/>
<aclentry name='OtherDomainServers' type='servergroup' level='noaccess' readpublicdocs='false'
 writepublicdocs='false'/>
<aclentry name='CN=Joerg Zeig/O=Zebis' type='person' level='manager' deletedocs='true'/>
<aclentry name='LocalDomainServers' type='servergroup' level='manager' deletedocs='true'/>
```

ACL in DXL

Die so erzeugten Informationen können nun in weitere XML-Formate umgesetzt werden.

Betrachten wir unser Beispiel nochmals, dann handelt es sich um einen recht überschaubaren Datenbestand. Trotzdem ist die erstellte DXL schon recht komplex. In der Praxis ist es jedoch meist erforderlich, einen Extrakt aus dem Gesamtbestand einer Datei zu bilden und nur ausgewählte Datensätze für einen XML-Export zur Verfügung zu stellen. Würden wir einen XML-Export – wie oben beschrieben – durchführen, dann würden uneingeschränkt alle Informationen aus der Domino-Datenbank in DXL

umgesetzt werden. Alle Informationen bedeutet, dass neben den eigentlichen Datensätzen auch die Design-Elemente, die Zugriffslisten, Log-Einträge etc. exportiert würden.

Um die Menge der in DXL zu exportierenden Informationen einzuschränken, stellt Lotus die Klassen *NotesDocumentCollection* und *NotesNoteCollection* zur Verfügung. Diese beiden Klassen erlauben eine Selektion der zu exportierenden Informationen.

6.6.1.1.1 NotesDocumentCollection

Das nachfolgende Beispiel zeigt, wie die Exportinformationen allein auf die Dokumente einer Datenbank beschränkt werden können. Dabei wird der Basiscode aus dem vorhergehenden Beispiel verwendet und um die Statements erweitert, die die Selektion darstellen. Diese werden nach dem ursprünglichen Statement „Call stream.truncate“ eingefügt. Einzig und allein die Anweisung „Call exporter.SetInput(db)“ im ursprünglichen Code wurde durch die Anweisung „Call exporter.SetInput(dc)“ ersetzt. Damit wird angegeben, dass der Inhalt der DocumentCollection verarbeitet werden soll – und nicht mehr wie zuvor, der Inhalt der gesamten Datenbank.

Ferner wird der nachfolgende Block eingefügt:

```
REM Code für Einschränkung angeben
Dim dc As NotesDocumentCollection
Set dc = db.AllDocuments
If dc.Count = 0 Then
   Messagebox "Keine Dokumente gefunden",, "Keine Dokumente"
   Exit Sub
End If
```

Code für Selektion

Der gesamte Code für dieses Beispiel ist in der folgenden Abbildung dargestellt.

```
Sub Click(Source As Button)
   Dim session As New NotesSession
   Dim db As NotesDatabase
   Dim pfadname As String
   Set db = session.CurrentDatabase
   pfadname$ = "c:\xmldemo\"

 REM Öffnen XML Datei
   Dim stream As NotesStream
   Set stream = session.CreateStream
   filename$ = pfadname$ &  Left(db.FileName, Len(db.FileName) - 3) & "xml"
   If Not stream.Open(filename$) Then
      Messagebox "Die Datei " & filename$ & "kann nicht geöffnet werden",, "Fehler!"
      Exit Sub
   End If
   Call stream.Truncate


   REM Code für Einschränkung angeben
   Dim dc As NotesDocumentCollection
   Set dc = db.AllDocuments
   If dc.Count = 0 Then
      Messagebox "Keine Dokumente gefunden",, "Keine Dokumente"
      Exit Sub
   End If


 REM Datenbank als XML exportieren
   Dim exporter As NotesDXLExporter
   Set exporter = session.CreateDXLExporter
   Call exporter.SetInput(dc)
   Call exporter.SetOutput(stream)
   Call exporter.Process
End Sub
```

Selektion auf Dokumente

Das Ergebnis dieses Scripts erzeugt wieder eine Datei XMLKunden01.xml. Diese enthält neben den generellen XML-Informationen nur die Datensätze der Notes-Datenbank. Unter anderem ist die Zugriffsliste in diesem Beispiel nicht enthalten. Dieses Beispiel könnte verwendet werden, um Daten mit Nicht-Notes-Anwendungen auszutauschen.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE database SYSTEM 'xmlschemas/domino_6_5.dtd'>
<database xmlns='http://www.lotus.com/dxl' version='6.5' replicaid='C1256E990036A599'
 path='XMLKunden01.nsf' title='XMLKunden01'>
<databaseinfo dbid='C1256E990036A599' odsversion='43' diskspace='327680' percentused='93.04
 numberofdocuments='5'><datamodified><datetime dst='true'>20040520T140101,44+02</datetime></
><datetime dst='true'>20040520T140859,18+02</datetime></designmodified></databaseinfo>
<document form='KundenstammXMLDemo'>
<noteinfo noteid='8fa' unid='5571D13C2EA3A8AAC1256E990036BD96' sequence='1'>
<created><datetime dst='true'>20040519T115754,78+02</datetime></created>
<modified><datetime dst='true'>20040519T115810,85+02</datetime></modified>
<revised><datetime dst='true'>20040519T115810,84+02</datetime></revised>
<lastaccessed><datetime dst='true'>20040519T115810,84+02</datetime></lastaccessed>
<addedtofile><datetime dst='true'>20040519T115810,84+02</datetime></addedtofile></noteinfo>
<updatedby><name>CN=Joerg Zeig/O=Zebis</name></updatedby>
<item name='KundenNr'><number>1</number></item>
<item name='Nachname'><text>Winzig</text></item>
<item name='Vorname'><text>Willi</text></item>
<item name='Strasse'><text>Hauptstrasse 47</text></item>
<item name='PLZ'><number>40000</number></item>
<item name='Ort'><text>Düsseldorf</text></item></document>
<document form='KundenstammXMLDemo'>
<noteinfo noteid='8fe' unid='1FE693864AE78118C1256E9A0041C6AC' sequence='1'>
<created><datetime dst='true'>20040520T135827,00+02</datetime></created>
<modified><datetime dst='true'>20040520T135948,34+02</datetime></modified>
<revised><datetime dst='true'>20040520T135948,33+02</datetime></revised>
<lastaccessed><datetime dst='true'>20040520T135948,33+02</datetime></lastaccessed>
<addedtofile><datetime dst='true'>20040520T135948,33+02</datetime></addedtofile></noteinfo>
<updatedby><name>CN=Joerg Zeig/O=Zebis</name></updatedby>
<item name='KundenNr'><number>2</number></item>
<item name='Nachname'><text>ITP Verlags GmbH</text></item>
<item name='Vorname'><text/></item>
<item name='Strasse'><text>Postfach</text></item>
<item name='PLZ'><text/></item>
<item name='Ort'><text>Kaufering</text></item></document>
<document form='KundenstammXMLDemo'>
```

XML-Dokumente

6.6.1.2 Import-Klasse

Das Gegenstück der Export-Klasse stellt die Import-Klasse (NotexDXLImporter) dar, welche ebenfalls seit Release 6 ausgeliefert wird. Vergleicht man die Import- und die Export-Klassen, dann fällt auf, dass die Import-Klasse wesentlich komplexer ist als die Export-Klasse. Der Grund dafür liegt in den vielfältigen Möglichkeiten der unterschiedlichen Varianten, mit denen Informationen in Domino importiert werden können.

Denn bei einem Import von Informationen in Notes/Domino müssen nicht nur die Daten betrachtet werden, sondern auch die speziellen Domino-relevanten Informationen – beginnend bei der Beschreibung der Datenbank, deren Eigenschaften, den ACLs und den Designelementen, um nur einige zu nennen.

Einige dieser Informationen können über fest definierte Parameter erfolgen, die für mehrere Bereiche genutzt werden können, andere müssen individuell angegeben werden.

Die Import-Klasse verfügt über einige Eigenschaften, die für jeden Import einzeln betrachtet und individuell eingesetzt werden sollten. Im Folgenden sollen diese kurz beschrieben werden. Bevor wir auf die einzelnen Eigenschaften näher eingehen, sehen wir uns die unterschiedlichen Parameterwerte an, die für viele der Eigenschaften verwendet werden können:

Op.Nr.	Option	Beschreibung
1	DXLIMPORTOPTION_IGNORE	Ignoriert die Information und nutzt bereits vorhandene Einträge in der Datenbank.
2	DXLIMPORTOTION_CREATE	Erzeugt neue Information im Zielsystem. Bereits bestehende Informationen bleiben unverändert.
3	Nicht verwendet	
4	Nicht verwendet	
5	DXLIMPORTOPTION_REPLACE_ELSE_IGNORE	Überschreibt vorhandene Informationen.
6	DXLIMPORTOPTION_REPLACE_ELSE_CREATE	Ersetzt bestehende Informationen und erzeugt bei Bedarf neue Einträge in der Datenbank aus dem XML-Dokument.
7	Nicht verwendet	
8	Nicht verwendet	
9	DXLIMPORTOPTION_UPDATE_ELSE_IGNORE	Bestehende Informationen werden aktualisiert. Existiert kein Eintrag zu den XML-Informationen, werden diese ignoriert.
10	DXLIMPORTOPTION_UPDATE_ELSE_CREATE	Bestehende Informationen werden aktualisiert und bei Bedarf neue Einträge mit den XML-Informationen erstellt.

Tabelle 1

Die einzelnen Optionen werden durch die Angabe der jeweiligen Optionsnummer im Source-Text verwendet. Diese Optionen werden Sie in den nachfolgenden Beispielen mehrfach wiederfinden. Achten Sie bei der Verwendung der Optionsnummern bitte darauf, dass diese bei der Programmierung als Textstring und nicht als Zahlen angegeben werden – anderenfalls wird die Eingabe falsch umgesetzt!

ACLImportOption
Mit XML können bei Bedarf ACL-Informationen in Domino eingestellt werden. Dies ist allerdings nicht in jedem Fall sinnvoll, denn mit diesem Verfahren können die Sicherheitseinstellungen von Notes umgangen werden. Deshalb sollte sehr genau überlegt werden, ob Aktualisierungen der Daten und Anwendungen in diesem Bereich durch XML-Importe wirklich sinnvoll sind.

Für die Verwendung dieser Eigenschaften können die in der Tabelle 1 angegebenen Parameter 1, 5, 9 und 10 verwendet werden:

- DXLIMPORTOPTION_IGNORE (1)
  - Default-Einstellung
  - ACL-Einstellungen von DXL werden komplett ignoriert und die Einstellungen der Domino-Datenbank genutzt
- DXLIMPORTOPTION_REPLACE_ELSE_IGNORE (5)
  - Die ACL-Einstellungen der Domino-Datenbank werden mit den Informationen aus der DXL-Datei überschrieben
- DXLIMPORTOPTION_UPDATE_ELSE_IGNORE (9)
  - Bestehende Einträge in der Domino-Datenbank werden durch Einträge aus der DXL-Datei ersetzt. Es werden keine neuen Einträge auf Basis der DXL-Informationen eingestellt.

- DXLIMPORTOPTION_UPDATE_ELSE_CREATE (10)
  - Bestehende Einträge in der Domino-Datenbank werden durch Einträge aus der DXL-Datei ersetzt. Neue Informationen aus der DXL-Datei werden in die Domino-Datenbank eingestellt.

Sprechen keine besonderen Gründe dafür, ist die Verwendung von DXLIMPORTOPTION_IGNORE zu empfehlen!

CreateFTIndex
Mit dieser Eigenschaft wird angegeben, ob für die Domino-Datenbank ein Volltextindex erstellt werden soll.

DesignImportOption
Über XML können neben den Dateninhalten auch Gestaltungselemente von Notes importiert werden. Über die Eigenschaft „DesignImportOption“ kann gesteuert werden, ob und wie diese Informationen in eine Domino-Datenbank importiert werden dürfen. Bei den Vergleichen der zu verwendenden Parameter ist zu berücksichtigen, dass Vergleiche zwischen den vorhandenen Gestaltungselementen und den zu importierenden Elementen wie folgt vorgenommen werden:

- gleicher Typ
- gleicher Name
- gleiche Sprache

Ein Vergleich ist nur dann zutreffend, wenn alle drei Vergleichsoptionen zutreffen!

Gültige Parameter sind:

- DXLIMPORTOPTION_IGNORE (1)
  - ist die Default-Einstellung
  - Es werden keine Design-Elemente aus DXL in die Domino-Datenbank importiert.
- DXLIMPORTOPTION_CREATE (2)
  - Erstellt neue Design-Elemente aus der DXL
- DXLIMPORTOPTION_REPLACE_ELSE_IGNORE (5)
  - Ersetzt Design-Elemente und ignoriert zu importierende, wenn diese nicht vorhanden sind.
- DXLIMPORTOPTION_REPLACE_ELSE_CREATE(6)
  - Ersetzt Design-Elemente und erstellt neue, wenn diese in der Domino-Datenbank nicht vorhanden sind.

ReplaceDbProperties
Mit dieser Eigenschaft kann angegeben werden, ob die Datenbankeigenschaften mit Informationen aus der zu importierenden XML-Datei überschrieben werden dürfen. Es empfiehlt sich in der Regel, die Default-Einstellungen für diese Option zu nutzen – diese steht nämlich auf „nicht importieren“!

ReplicaRequiredForReplaceOrUpdate
Mit dieser Option lässt sich in Verbindung mit entsprechenden Log-Einträgen sehr gut feststellen, welche Informationen und Dokumente aus der XML-Import-Datei in Domino erfolgreich importiert werden konnten und welche Informationen aufgrund eines Fehlers während des Import-Vorgangs nicht berücksichtigt worden sind. Diesen Parameter sollte man auf der Default-Einstellung (diese steht auf „aktiviert“) belassen.

UnknownTokenLogOption
Diese Option legt in Verbindung mit den übrigen protokollrelevanten Optionen fest, wie die Protokoll- oder auch Log-Dateien gefüllt werden sollen. Dabei wird nicht der Inhalt der Informationen geprüft, sondern ein Abgleich zwischen den zu importierenden Dokumenten gegen die vorhandene oder auch mitgelieferte Struktur der Domino-Datenbank vorgenommen!

Der Entwickler kann hier zwischen den nachfolgenden Einstellungen unterscheiden. Überlegen Sie genau, welche Option sie für den Import ihrer Daten verwenden wollen – in der Praxis hat sich der Einsatz von DXLLOGOPTION_FATALERROR in Verbindung mit DXLLOGOPTOPN_IGNORE durchaus bewährt. (Letztere allerdings nur bei Eingabe-Validierungs-Prüfungen während der DTD-Verarbeitung!)

- DXLLOGOPTION_FATALERROR
  - Prüfungen werden mit einem schweren Fehler abgeschlossen und entsprechend mit diesem Vermerk in die Log-Datei geschrieben – der Import-Vorgang wird abgebrochen
- DXLLOGOPTION_ERROR
  - Prüfungen werden als „fehlerhaft“ abgeschlossen und es wird ein entsprechender Eintrag in der Log-Datei geschrieben – der Import-Vorgang wird aber nicht abgebrochen
- DXLLOGOPTION_WARNING
  - Prüfungen werden mit einem Warnhinweis in der Log-Datei protokolliert. Die Verarbeitung der geprüften Informationen wird aber nicht abgebrochen, sondern unbeeinflusst fortgesetzt.

- DXLLOGOPTION_IGNORE
  - Prüfungen werden nicht in der Log-Datei fortgeschrieben. Diese Option sollte zum Beispiel bei Eingabe-Validierungen, die über die DTD abgearbeitet werden sollen, gewählt werden.

DocumentImportOption
Für den Import-Vorgang ist dies sicher die wichtigste Funktion, denn mit ihr wird unter anderem angegeben, welche Dokumente beim Import berücksichtigt werden sollen. Für diese Option sind die in Tabelle 1 angegebenen Optionsnummern 1, 2, 5, 6, 9 und 10 verwendbar.

Für die Kontrolle der Importoption bietet sich die Verwendung von *ImportNoteCount*, mit der die Anzahl der eingelesenen Dokumente mitgezählt wird. Dies ist zu Kontrollzwecken sicher eine sinnvolle Funktion!

Sie können bei der Import-Funktion selbst entscheiden, ob die in der zu importierenden Datei mitgesendete DTD verwendet werden soll. Über diese DTD wird dann der Aufbau des zu importierenden Dokuments entsprechend geprüft.

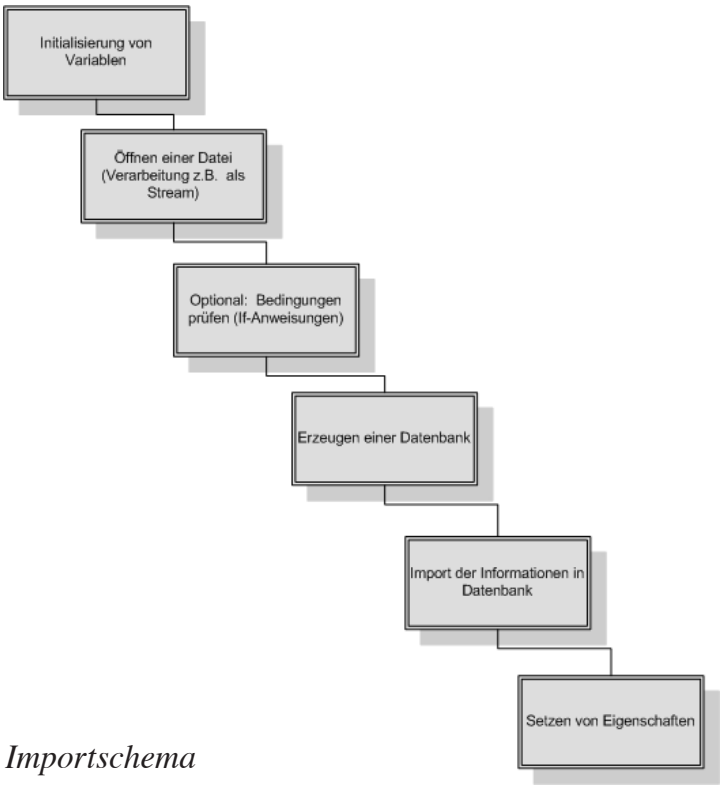
Für diese Funktion gibt es drei unterschiedliche Auswahlen:

- VALIDATE_NEVER
  - Es wird keine DTD-Prüfung durchgeführt
- VALIDATE-ALWAYS
  - Es wird immer eine entsprechende DTD-Prüfung durchgeführt. Schlägt diese fehl oder wird keine entsprechende DTD gefunden, dann wird eine Fehlermeldung ausgegeben und der Import abgebrochen.

- VALIDATE_AUTO
  - Eine Prüfung der zu importierenden Dokumente wird mit Hilfe der vorhandenen DTD durchgeführt. Existiert keine DTD, dann wird keine Prüfung durchgeführt.

Nachdem wir nun die Optionen und einige Voraussetzungen für den Import von Informationen mittels XML in Domino besprochen haben, sollten wir uns einmal anschauen, wie der Import praktisch umgesetzt werden könnte.

Die nachstehende Abbildung soll dies verdeutlichen:



Importschema

Der Import von Informationen mittels XML läuft im Groben eigentlich genau nach diesem Schema ab.

Die nachfolgende Beschreibung soll anhand eines Muster-Code-Ausschnitts zeigen, wie die entsprechenden Schritte für den Import in einem Programmgerüst abgebildet werden können.

Hier ist anzumerken, dass es sich auch im Bereich der XML/Domino-Anwendungsentwicklung anbietet, auf bestimmte Code-Gerüste zurückzugreifen, die man für die einzelnen Anforderungen entsprechend anpassen kann. Ein solches Grundgerüst könnte der beigefügte Beispielcode darstellen. Doch erst einmal weiter mit der Import-Funktion:

Dabei werden zunächst die erforderlichen Variablen initialisiert, um anschließend eine Datei zu öffnen. Das Verarbeiten dieser Datei kann auf unterschiedliche Arten und Weisen erfolgen, in der Praxis hat sich die Verarbeitung als „Stream“ durchaus bewährt. In dem unten angeführten Beispielcode wird die Datei auch entsprechend als „Stream“ verarbeitet.

Nachdem die Datei eröffnet wurde, können entsprechende Bedingungen abgefragt werden. Dies kann optional geschehen, was in der Praxis durchaus gängig ist.

Nach dem Durchführen der optionalen Bedingungen wird eine leere Datenbank erzeugt. Diese dient für den Import der Informationen. Nachdem diese Datenbank erzeugt wurde, wird zunächst ein Objekt für den DXL-Import generiert.

Um die Erstellung der Datenbank abzuschließen, müssen die entsprechenden Eigenschaften gesetzt werden. Diese wurden weiter oben näher beschrieben.

Sind all diese Schritte durchgeführt worden, kann der eigentliche Import der Informationen erfolgen.

Das hört sich im ersten Moment alles vielleicht ein wenig verwirrend an – aber schauen Sie sich den Beispielcode einmal in Ruhe an – diesen können Sie als Grundgerüst für den XML-Import durchaus verwenden.

Programmtext

```
Sub Initialize
 REM Initiliaisieren der Variablen
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim dbCopy As NotesDatabase
    Set db = session.CurrentDatabase
    filename$ = Left(db.FileName, Len(db.FileName) - 4)

 REM Verarbeitung als "Stream" festlegen
    Dim stream As NotesStream
    Set stream = session.CreateStream
    If Not stream.Open("c:\dxl\" & filename$ & ".dxl") Then
       Messagebox "Cannot open " & filename$,, "Error"
       Exit Sub
    End If
    If stream.Bytes = 0 Then
       Messagebox "Datei existiert nicht oder war leer!",, filename$
       Exit Sub
    End If

    Set dbCopy = New NotesDatabase ("", "")
    Call dbCopy.Create("", filename$ & "Copy", True)


 REM Ersetzen der Dokument in der Datenbank durch DXL Import-Dokumente
    Dim importer As NotesDXLImporter
    Set importer = session.CreateDXLImporter(stream, dbCopy)
    importer.ReplaceDBProperties = True
    importer.ReplicaRequiredForReplaceOrUpdate = False
    importer. ACLImportOption = DXLIMPORTOPTION_REPLACE_ELSE_IGNORE
    importer.DesignImportOption = DXLIMPORTOPTION_CREATE
    importer.DocumentImportOption = DXLIMPORTOPTION_REPLACE_ELSE_IGNORE
    Call importer.Process

End Sub
```

Import Beispiel Script

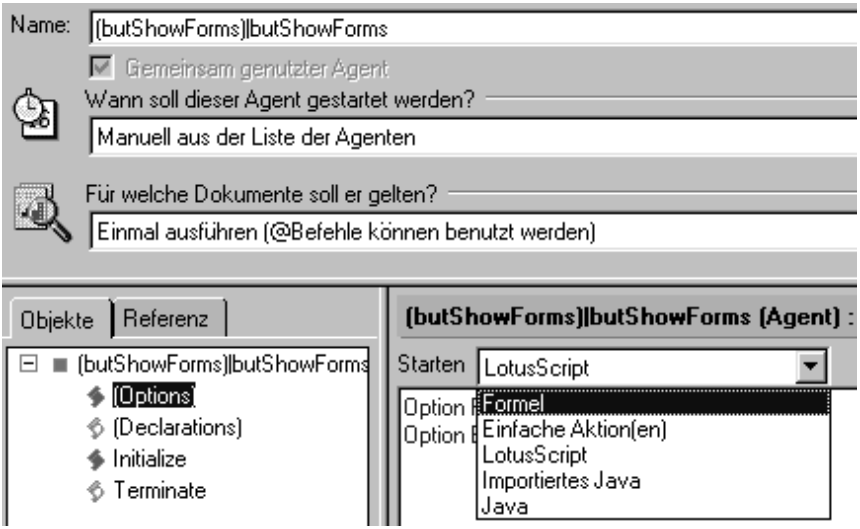
Mit diesem einfachen Beispiel haben wir nun die Möglichkeit geschaffen, Daten mittels XML in Domino zu importieren. Dieses Beispiel ist mehr als Einstieg zu betrachten – denn komplexe Umgebungen bedingen zum Teil auch ebenso komplexe Importfunktionen der Daten. Neben den in diesen Abschnitt gezeigten Import-Funktionen und den dafür gezeigten Werkzeugen gibt es noch einige weitere Möglichkeiten, wie Informationen in Notes/Domino mit XML importiert und exportiert werden können bzw. wie die Import- und Exportfunktionen individuell angepasst und optimiert werden können. Diese Möglichkeiten werden wir in den nächsten Abschnitten näher betrachten.

Diese Seite ist zur Zeit nicht belegt.

7

Agenten

Agenten sind Programme, die meist zur Durchführung umfangreicherer Aktionen verwendet werden. Sie können als Teil eines Ereignisses oder auch zeitplangesteuert ihre Aufgaben erfüllen.



Der Editor zum Bearbeiten von Agenten

Der Einsatz von Agenten, die über einen Zeitplan oder ein bestimmtes Ereignis aktiviert werden, erlaubt es, bestimmte Überwachungs- und Wartungsarbeiten in einem geplanten Umfang durchführen zu lassen. In einer der folgenden Programmiersprachen können Agenten geschrieben werden:

- Einfache Aktionen
- Formelsprache
- LotusScript
- Importiertes Java
- Java

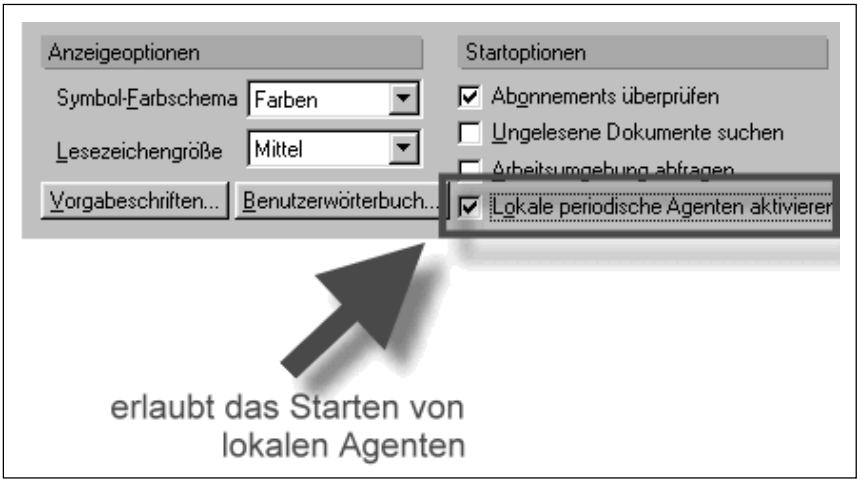
Damit bieten sich hier alle Möglichkeiten vollständiger Programmiersprachen, die eine komplexe Verarbeitungen und Datenzugriffe auf verschiedenen Datenquellen erlauben.

Bei dem Wort Programmiersprachen könnte man denken, dass es sich hier um ein Werkzeug für Programmierer handelt. Das ist aber durchaus nicht der Fall. Wenn der Benutzer seine Arbeitsumgebung entsprechend eingerichtet bekommt, ist er in der Lage, einfache Agenten selbst zu erstellen und auch auszuführen.

Diese Seite ist zur Zeit nicht belegt.

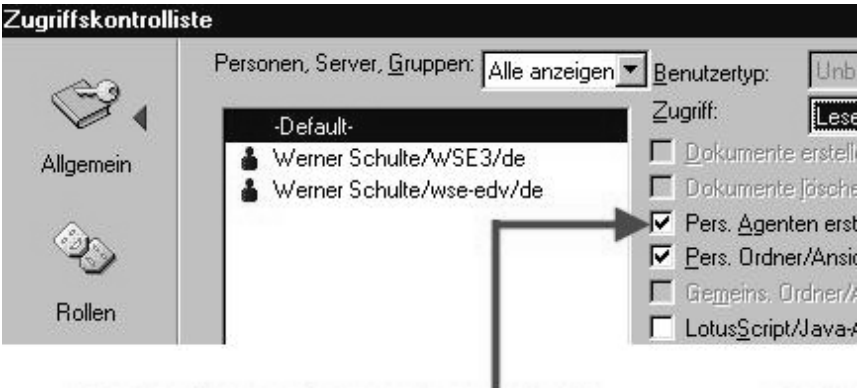
7.1 Lokale Agenten

Lokale Agenten, also solche, die auf der Arbeitsstation des Benutzers ausgeführt werden sollen, sind dadurch gekennzeichnet, dass sie nur durch eine Benutzeraktion gestartet werden können. Auf einem Server ist der Agenten-Manager aktiv, der alle Datenbanken auf ihre Agenten hin untersucht und sie gemäß der Parameter für die Agentensteuerung ausführt. Die lokale Ausführung von Agenten ist **nicht** generell gestattet. Die erste Hürde ist in den Benutzervorgaben zu nehmen. Im Menü „Datei/Benutzervorgaben“ ist im allgemeinen Teil des Dialoges ein Kontrollfeld einzuschalten, damit lokale Agenten gestartet werden dürfen.



Benutzervorgaben - lokale Agenten

Eine weitere Hürde stellt die Berechtigung des Benutzers an der Datenbank dar, die den Agenten aufnehmen soll. In der Zugriffskontroll-Liste ist hinterlegt, welche Benutzer bzw. Benutzergruppen Agenten aktivieren dürfen. Dabei geht Domino zweistufig vor. Zunächst wird über die Berechtigungsstufe definiert, ob der Benutzer die Rechte „Manager", „Designer" usw. haben soll. Nur wenn seine Rechte unter denen eines Designers eingestuft sind, kann das Recht, persönliche Agenten zu starten, einzeln aktiviert bzw. deaktiviert werden.

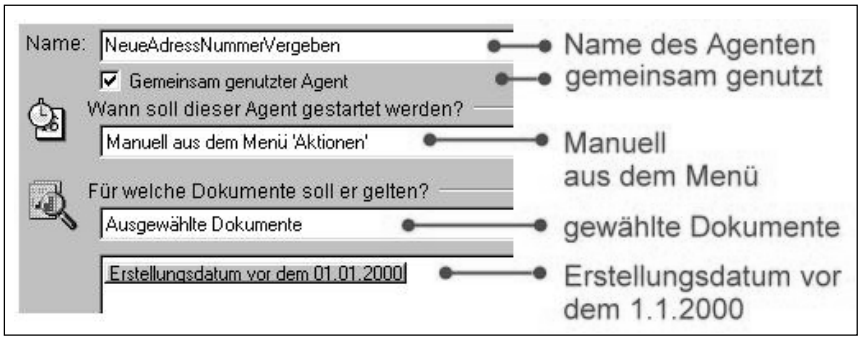


Persönliche Agenten in der Zugriffskontroll-Liste

Sind von dieser Seite her die Voraussetzungen für die Erstellung des Agenten gegeben, so beginnt jetzt die eigentliche Erstellung des Agenten.

Im Menü „Erstellen" wird unter „Gestaltung" die Position „Agent" ausgewählt. Es öffnet sich die Entwicklungsumgebung für Agenten. Damit der Agent auch nach seiner Speicherung wieder auffindbar ist, bekommt er zu allererst einen Namen. Der Name kann

zwar beliebig sein, sollte aber möglichst nahe an seiner Funktion liegen. Namen wie „James Bond“ oder „007“ sind wahrscheinlich sehr aufregend, in der Praxis aber wenig hilfreich. Sollen z. B. Adressen neu numeriert werden, so wäre „NeueAdressNummerVergeben“ ein sinnvoller Name.



Agent erstellen

Als nächste Eigenschaft des Agenten ist festzulegen, ob er gemeinsam benutzbar sein soll oder nicht. Die Wirkung dieser Eigenschaft besteht darin, dass der Agent entweder in der Desktop.dsk auf der Arbeitsstation oder in der Datenbank gespeichert wird. Nur ein gemeinsam benutzter Agent wird in der Datenbank gespeichert. Aus diesem Grund kann diese Eigenschaft des Agenten nicht mehr verändert werden, wenn sie einmal festgelegt ist.

Agent gemeinsam nutzen

In den Optionen lässt sich definieren, ob die Suche im Suchleistenmenü angezeigt, ob Markierungen in den Dokumenten gespeichert und ob die Dokumente für Benutzer mit öffentlichem Zugriff verfügbar sein sollen.

Die Frage, wann der Agent gestartet werden soll, ist zunächst einmal mit der Beschreibung der Art und Weise zu beantworten, wie das erfolgen soll. Dafür werden folgende Möglichkeiten angeboten:

- manuell aus dem Menü „Aktionen“

  Der Agent kann aus dem Menü manuell gestartet werden.

- manuell aus der Liste der Agenten

  Der Agent ist für den Benutzer nicht sichtbar. Er kann aber z.B. per Programm gestartet werden.

- vor Eingang einer neuen Mail

  Der Agent wird automatisch gestartet, wenn eine neue e-Mail eingegangen, dem Empfänger aber noch nicht zugestellt worden ist.

- nach Eingang einer neuen Mail

  Der Agent wird automatisch gestartet, wenn eine neue e-Mail an den Empfänger zugestellt worden ist.

- wenn Dokumente erstellt oder geändert wurden

  Der Agent wird automatisch gestartet, wenn in der Datenbank ein Dokument erstellt oder geändert wurde.

- wenn Dokumente eingefügt werden

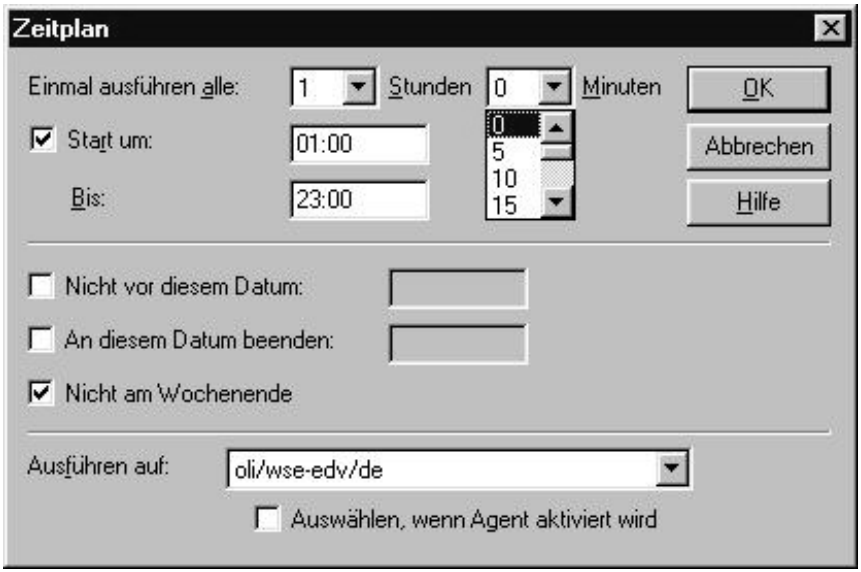
  Der Agent wird automatisch gestartet, wenn ein Dokument in die Datenbank eingefügt wurde.

- periodisch: mehrmals am Tag
- periodisch: täglich
- periodisch: monatlich

  Der Agent wird automatisch gestartet, wenn er nach Zeitplandefinition aktiv werden soll.
- periodisch: nie

  Der Agent wird in keinem Fall gestartet. Solange diese Option eingestellt ist, ist der Agent deaktiviert.

Für den Fall, dass eine periodische Aktivierung des Agenten erfolgen soll, muss ein Zeitplan erstellt werden. Über die Schaltfläche „Zeitplan“ wird der dazugehörige Dialog geöffnet:



Agent – Zeitplan

Im Zeitplandialog sind folgende Angaben möglich:

- das Intervall zwischen den Startterminen des Agenten (alle ... Stunden, Minuten)
- die Tageszeit
- der x-te Tag der Woche oder des Monats
- ein Startdatum
- ein Enddatum
- Ausführung am Wochenende

Außerdem kann hier der Server festgelegt werden, auf dem der Agent aktiv werden soll. Wird die Option „Beliebiger Server“ genutzt, so ist darauf zu achten, dass der Agent nicht mehrere Repliken der Datenbanken auf verschiedenen Servern bearbeitet. Das könnte zu umfangreichen Speicher- und Replizierkonflikten führen. Sinnvoll ist es, den Server erst zur Laufzeit festzulegen. Das erfolgt über das Kontrollfeld: „Auswählen, wenn Agent aktiviert wird“. Dabei ist allerdings darauf zu achten, dass eine Schaltfläche oder ein Menüeintrag zur Aktivierung des Agenten vorgesehen wird.

Um festzulegen, welche Dokumente von dem Agenten verarbeitet werden, kann je nach Aktivierungsart aus allen oder einzelnen der nachstehenden Möglichkeiten ausgewählt werden:

- alle ungelesenen Dokumente in der Ansicht

  Der Agent verwendet alle Dokumente, die vom aktuellen Benutzer noch nicht gelesen wurden.

- alle Dokumente der Ansicht oder der Datenbank

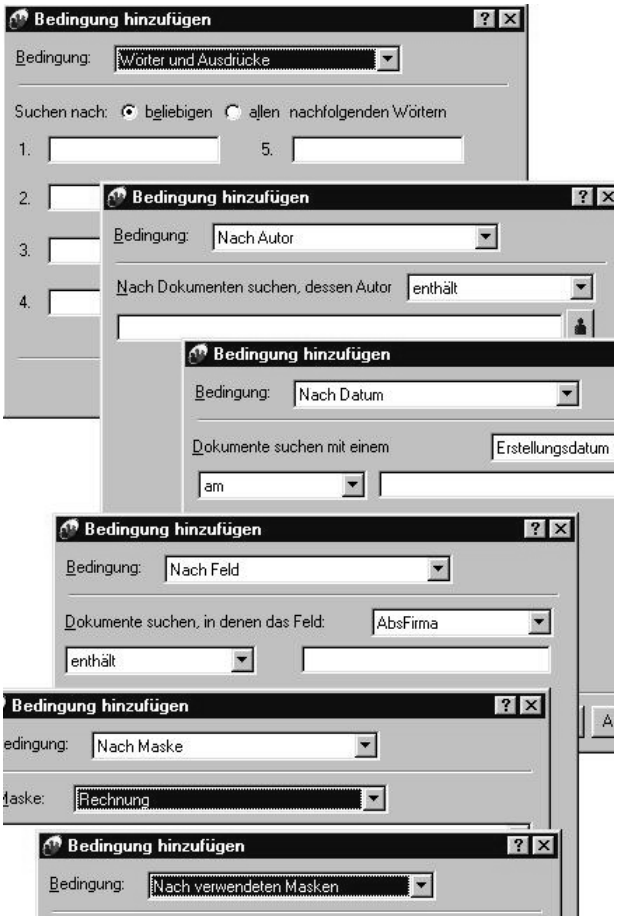
  Der Agent verwendet alle Dokumente, die sich in der aktuellen Ansicht oder der Datenbank befinden.

- ausgewählte Dokumente

  Der Agent verwendet alle ausgewählten Dokumente.

- Einmal ausführen (@-Befehle können verwendet werden)
  Der Agent wird über ein Programm gestartet.

Die Auswahl der zu verarbeitenden Dokumente kann auch über einen speziellen Dialog zur Definition der Suche getroffen werden:



Agenten-Suche definieren

Der Dialog ändert sich nach der Angabe der Bedingung und erfragt in Abhängigkeit von dieser Bedingung weitere Angaben. Als Bedingung stehen zur Verfügung:

- Wörter und Ausdrücke

  Wörter und Ausdrücke, die in den gesuchten Dokumenten enthalten sein müssen.

- nach Autor

  Personen, die in einem Autorenfeld der gesuchten Dokumente enthalten sein sollen.

- nach Datum

  Dokumente werden gewählt, die in einem bestimmten Zeitabschnitt geändert oder erstellt wurden.

- nach Feld

  Die Dokumentauswahl erfolgt anhand bestimmter Feldwerte.

- nach Maske, bzw. verwendeter Maske

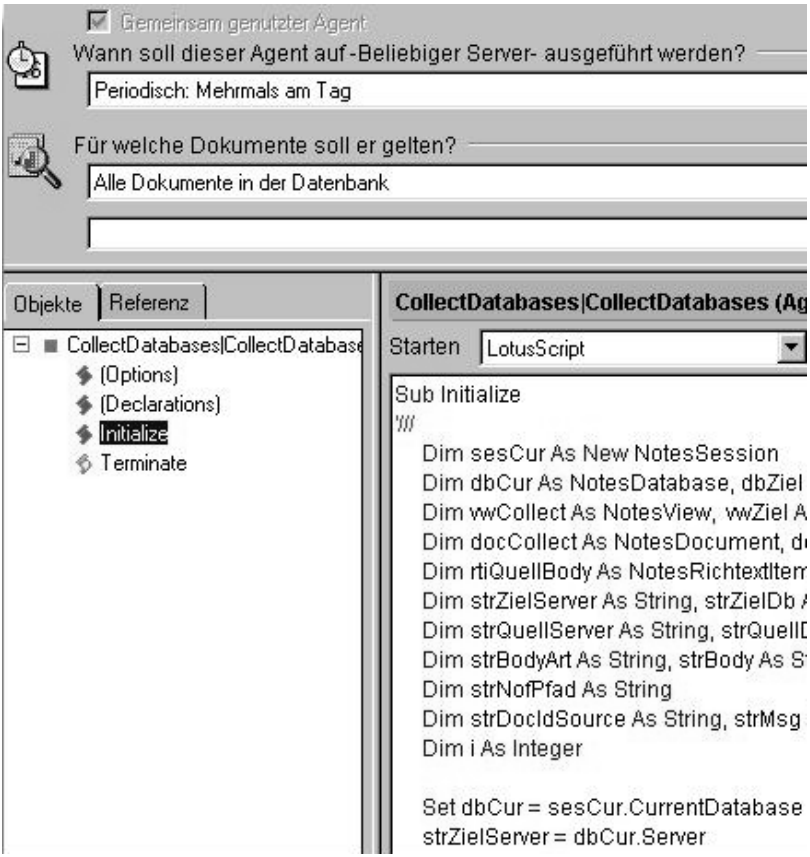
- nach Ordner

Ist jetzt endgültig festgelegt, wann der Agent ausgelöst wird und auf welche Dokumente er sich bezieht, so wird als letztes bestimmt, welche Aufgabe der Agent ausführen soll. Die Ausführungsdefinition stellt ein Programm dar, das mit verschiedenen Methoden erstellt werden kann. Zur Programmierung stehen zur Verfügung:

- die Formelsprache von Lotus
- „Einfache Aktionen“ (eine Definition über Benutzerdialoge)

- LotusScript, der Basic-Dialekt von Lotus
- Java

Für einen geschulten Benutzer kommen die „Einfachen Aktionen“ und eventuell die Formelsprache in Betracht. LotusScript und Java sind Programmiersprachen, die ein umfangreiches Entwickler-Know-how erfordern.



LotusScript-Agent

Es sollen deshalb an dieser Stelle nur einige der „Einfachen Aktionen“ aufgeführt werden:

- In Datenbank kopieren

  Die ausgewählten Dokumente werden in eine angegebene Datenbank kopiert.

- In Ordner kopieren

  Der Agent stellt die ausgewählten Dokumente in einen festgelegten Ordner.

- Feld ändern

  Ändert in den gewählten Dokumenten einen Feldwert.

- Dokument senden

  Versendet die ausgesuchten Dokumente.

7.2 Server-Agenten

Sollen Agenten auf einem Server ausgeführt werden, so sind weitere Berechtigungsschritte erforderlich. Im Server-Dokument wird angegeben, wer berechtigt ist, einfache oder unbeschränkte Agenten auf einem Server auszuführen. Solche Agenten sind häufig nach einem Zeitplan gesteuert und verwenden umfangreichere Programme. Server-Agenten können auch aus lokalen Agenten und Aktionen heraus gestartet werden.

Diese Seite ist zur Zeit nicht belegt.

8

Externe Datenquellen

Der Zugriff auf externe Daten ist in Lotus Domino über verschiedene Methoden und mit unterschiedlichen Hilfsmitteln möglich. Einfache Möglichkeiten sind zum Beispiel die Verwendung von Notes-Field-Exchange und die Verarbeitung von externen Textdateien in LotusScript.

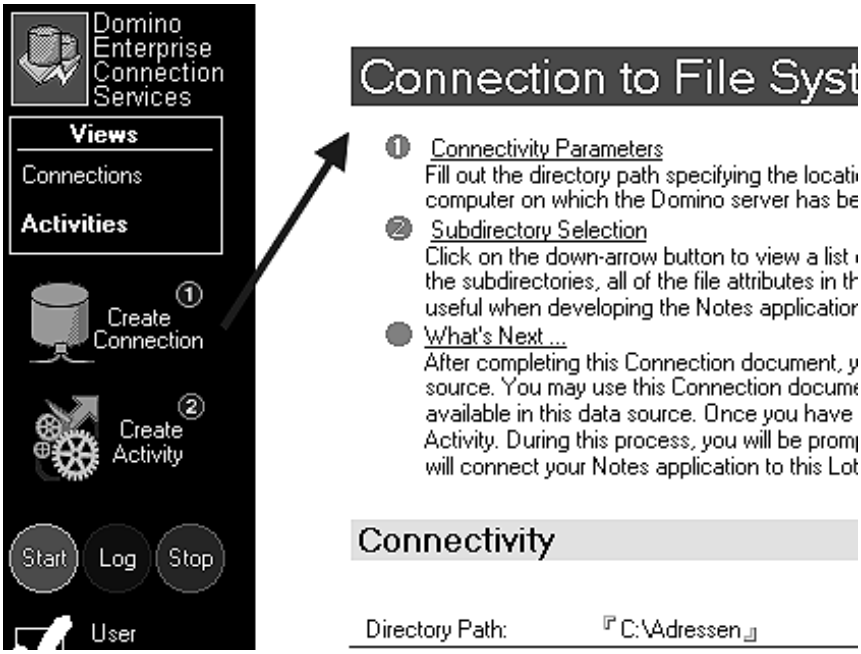
Die gängigste Zugriffsmethode stellt zur Zeit noch ODBC dar. Lotus stellt für den Zugriff auf Notes-Daten einen NotesSQL genannten Treiber für ODBC zur Verfügung.

Über ODBC-Datenquellen können die Formeln @DbColumn, @DbLookUp und @DbCommand auf die Daten relationaler Datenbanken zugreifen und Werte oder Listen von Werten übernehmen.

Mit LS:DO, einer Ergänzung zu LotusScript, ist ebenfalls ein ODBC-Zugriff möglich. Die Klassen ODBCConnection, ODBCQuery und ODBCResultSet erlauben eine erweiterte Verarbeitung externer Daten.

Über den Einsatz der Programmiersprache Java und die Nutzung der Klassen zur Verarbeitung relationaler Datenbanken (JDBC) bietet Domino einen weiteren Weg zu externen Daten.

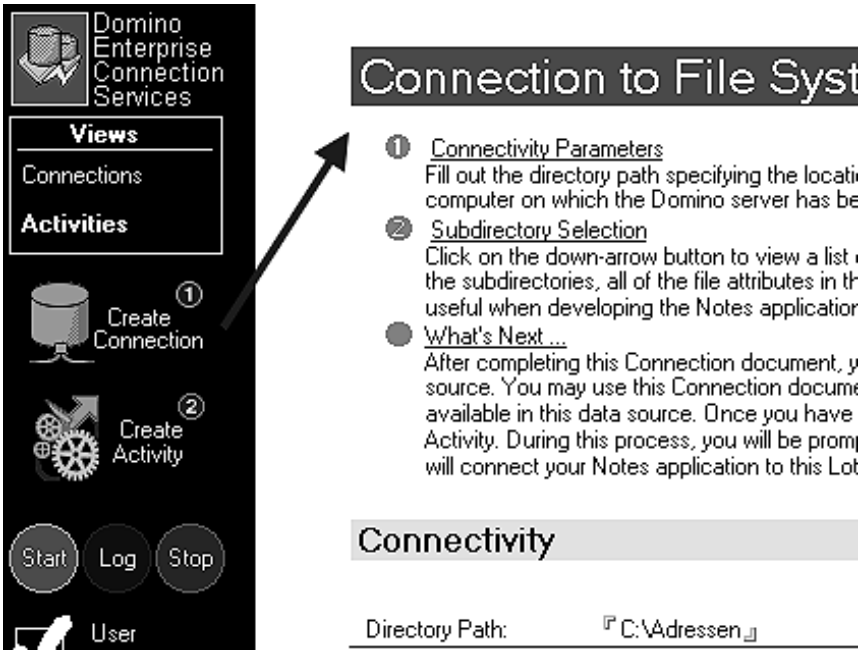
Mit Release 5 wurden die Domino Enterprise Connection Services (DECS) eingeführt. Dieses System beinhaltet einen Server-Prozess für Echtzeit-Datenverbindungen, ein Administrations-Tool (DECS Administrator) und eine Datenbank (DECS Administrator).



DECS-Administrator

Für temporäre Zugriffe werden die ODBC-Funktionen und Formeln, aber ein eigenes Treiber-Modell, die Domino Connectors, verwendet. Diese Verbindungsmodule sind nicht nur für relationale Datenbanken wie DB2, Oracle und Sybase, sondern auch für ERP-Systeme wie SAP und PeopleSoft verfügbar.

Diese Seite ist zur Zeit nicht belegt.

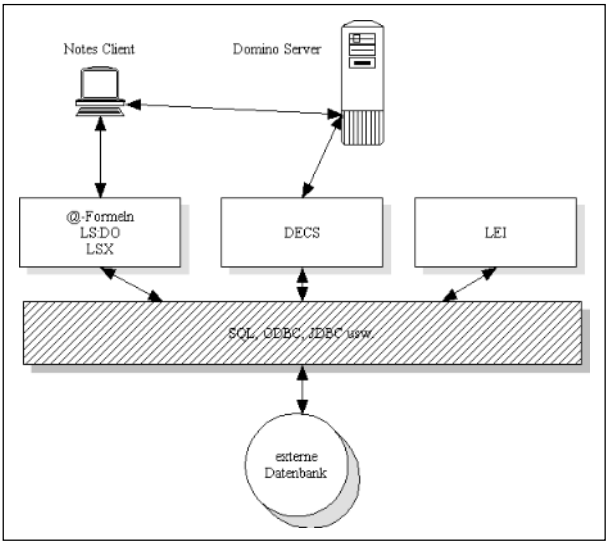


DECS-Administrator

Für temporäre Zugriffe werden die ODBC-Funktionen und Formeln, aber ein eigenes Treiber-Modell, die Domino Connectors, verwendet. Diese Verbindungsmodule sind nicht nur für relationale Datenbanken wie DB2, Oracle und Sybase, sondern auch für ERP-Systeme wie SAP und PeopleSoft verfügbar.

Der Zugriff auf externe Daten kann über den Client oder den Server erfolgen. Beim Zugriff durch den Client müssen die entsprechenden Verbindungen zu den Datenbanken auf dem Rechner des Clients zur Verfügung stehen (z. B. ODBC). Der Zugriff über den

Server kann durch DECS realisiert werden. Die verschiedenen Zugriffsarten zeigen folgendes Schaubild:



Zugriff auf externe Datenbanken

Als Datenbankquellen unterstützt Lotus Domino Verbindungen zu folgenden Datenbankarten:

- IBM DB2
- EDA/SQL
- Open Database Connectivity (ODBC)
- Oracle
- Sybase
- Zmerge Text

Dazu werden von den einzelnen Herstellern der Datenbanken unterschiedliche Verbindungsprogramme geliefert, die wiederum von der jeweiligen Hardwareplattform abhängig sind.

8.1 Domino Enterprise Connection Services (DECS)

Mit Release 4.6.3 von Lotus Domino wurde ein Teil der Datenübertragungssoftware NotesPump als DECS in Domino übernommen. Damit ist es möglich, Datenquellen außerhalb von Domino anzusprechen, ohne Programmiersprachen einsetzen zu müssen. DECS besteht aus einem Server-Prozess und einer Datenbank mit Dokumenten für Verbindungen und Aktivitäten. Während Lotus Enterprise Integrator (LEI), das frühere NotesPump, unterschiedliche Formen des Datenabgleichs und -austausches zur Verfügung stellt, lässt sich mit DECS ausschließlich der direkte Zugriff auf externe Datenbanken wie DB2, Oracle u.a. realisieren. Ist eine Datenquelle über DECS definiert, kann in Notes-Datenbanken direkt oder über die verschiedenen Programmiersprachen darauf zugegriffen werden.

DECS besteht aus zwei Komponenten:

- Management Applications

  Unter Management Applications versteht man eine Domino-Datenbank, die mindestens aus einer Ansicht und einem Feld besteht, um die Daten der externen Datenbank aufzunehmen (Verbindungen). Die Datenbank wird darüber hinaus dazu verwendet, definierte Operationen (Aktivitäten) auszuführen. Sie wird automatisch bei der Installation erstellt, wenn man DECS als Option ankreuzt.

- Server-Add-In-Prozess

  Der Server-Add-In-Prozess stellt die Verbindung zwischen den Definitionen in den Management Applications und dem Domino Server Extension Manager her. Über diese Methode werden die DECS-Funktionen als Server-Programm ausgeführt. Der DECS-Prozess stellt sowohl Notes Clients als auch Browsern die externen Daten bereit. Der Client muss also keine direkte Datenverbindung herstellen. Umgekehrt muss der Server mit den entsprechenden Datenbanken netzwerktechnisch verbunden sein. Dieses Konzept schränkt die Nutzung von DECS auch ein. Ein Client, der nicht mit einem Server verbunden ist, hat dadurch keinen Zugriff auf externe Daten über DECS.

8.1.1 Voraussetzungen

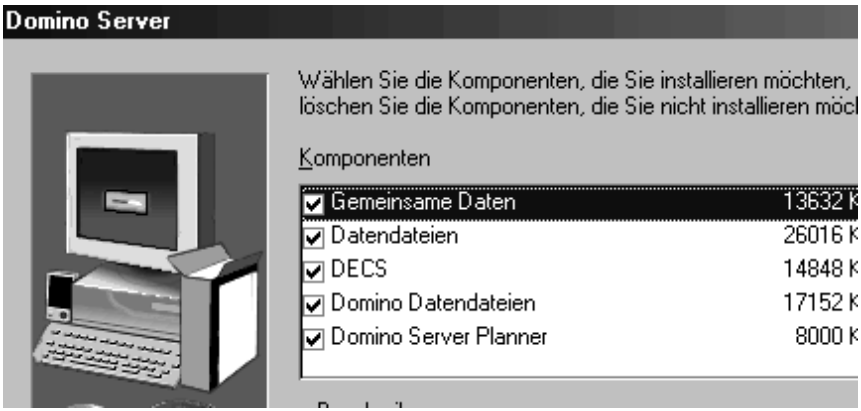
Nachfolgende Voraussetzungen müssen erfüllt sein, um DECS einsetzen zu können:

- Lotus Domino ab Version 4.6.3
- Domino Designer, um Notes-Datenbanken für die Verknüpfung von Notes zu externen Datensystemen zu definieren
- Kenntnisse in der Programmierung von Lotus Notes (Erstellen einer Datenbank, Erstellen einer Ansicht, Erstellen einer Maske)
- Ausreichende Berechtigung für den Zugriff auf die externe Datenbank (es werden Lese- und Schreibrechte benötigt)
- Ausreichende Berechtigung in Lotus Notes
- Zugriff auf ein von DECS unterstütztes externes Datenbanksystem
- Kopplungssoftware, die das Bindeglied zwischen Notes und den externen Daten bildet, beispielsweise im Fall eines Zugriffes von Lotus Notes auf DB2/400 wird die Software DB2 Connect Personal Edition benötigt. Diese ist frei aus dem Internet herunterladbar. Da die WEB-Version jedoch mehrfach Probleme bereitet hat und trotz OK-Meldung des Ladevorganges offensichtlich nicht ganz vollständig vom WEB geladen wurde, wird empfohlen, die DB2 Private Edition bei IBM zu bestellen. Diese Software ist derzeit kostenlos erhältlich. Es empfiehlt sich auf den WEB-Seiten von Lotus über die erforderlichen Sofwaretools für die unterschiedlichen Plattformen Informationen aktuell einzuholen.

Diese Seite ist zur Zeit nicht belegt.

8.1.2 DECS-Installation

DECS kann im Zuge der Domino-Installation bzw. im Zuge eines Upgrades von einem Domino Server installiert werden. Wird ein Domino Server neu installiert und die Standardinstallation ausgewählt, so erfolgt automatisch auch das Einspielen der DECS-Funktionen. Wird der Domino Server benutzerdefiniert installiert, so muss für die Installation der DECS-Funktionen die Funktionsgruppe DECS ausgewählt werden.



DECS-Installation

Während des Installationsvorganges werden verschiedene DECS-Dateien auf dem Domino Server erstellt. Unter anderem wird das DECS-Administrations-Template angelegt (DECSADM.NTF) und die komplette Dokumentation der DECS-Funktionen im „Help“-Verzeichnis erstellt.

Waren in der Version 4.6.3 noch diverse Einstellungen in der Datei „NOTES.INI“ notwendig, so hat sich mit der Version 5.0.2 die Installation in diesem Bereich stark vereinfacht. Im Normalfall sind keine manuellen Anpassungen an der „NOTES.INI“ erforderlich.

Nachfolgend die Einstellungsparameter in der „NOTES.INI“ (nur notwendig bei Versionen kleiner 5.0.2):

- DECSTranslation
  steuert die Zeichenumsetzung von Notes zu externen Daten
- DECSNativeText
  ermöglicht das Überschreiben des Zeichensatzes
- DECSCenturyBoundary
  steuert die Datumsumsetzung/Datumsdarstellung in Notes
- EXTMGR_ADDINS
  Dieser Eintrag ist notwendig, wenn parallel neben DECS auch noch LEI auf dem Domino Server läuft.

8.1.3 Datenquellen definieren

Wird eine Datenaustauschmethode im Notes-Umfeld gesucht, die weitgehend ohne spezielle Programmierkenntnisse auskommt und die im stationären Betrieb eingesetzt werden soll, dann bietet sich der Einsatz von DECS an. Man muss sich aber darüber im Klaren sein, dass Kenntnisse über den Aufbau der Datenbanken erforderlich sind. Ganz ohne Programmierkenntnisse kommt man auch mit DECS nicht aus, denn in Notes muss für den Datenaustausch eine eigene Datenbank (inklusive Ansicht, Maske und Feldern) erstellt werden. Ziel dieser Domino-Datenbank ist es, Felder zur Verfügung zu stellen, die im Weiteren als Aufnahmefelder für die Datenfelder der externen Datenbanken dienen.

Folgende Arbeitsschritte sind notwendig, um DECS als Datenaustauschmethode zu verwenden:

- Erstellen einer Notes-Datei inkl. Ansicht, Maske und Feldern
- Erstellen eines Verbindungsdokuments für die externe Datei
- Erstellen einer Real-Time-Activity für den Datenaustausch zwischen Notes und der externen Datei

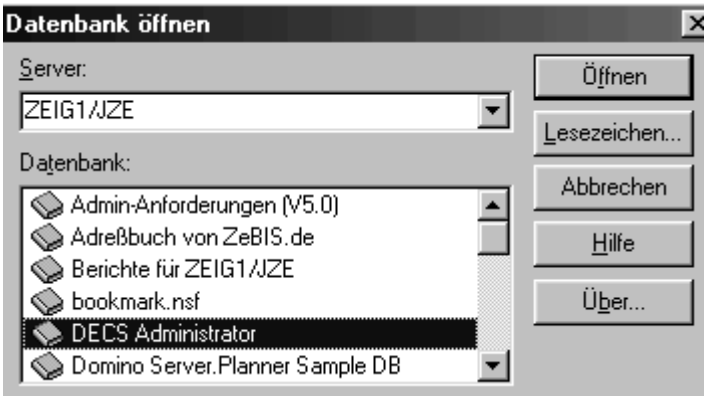
Die Verwaltung der Aktivitäten wird über den DECS Administrator vorgenommen. Der DECS Administrator ist Bestandteil des Administrator-Clients und wird dort nur dann angezeigt, wenn DECS auch installiert ist.

Diese Seite ist zur Zeit nicht belegt.

8.1.4 Der DECS Administrator

Der DECS Administrator (nicht zu verwechseln mit dem Domino Administrator Client!) ist eine Notes-Datenbank, in der die Verbindungen zu externen Datenbanken definiert werden können. Alle DECS-Funktionen können aus dieser Datei heraus bedient und eingestellt werden.

Der DECS Administrator kann direkt von einem Notes Client aus aufgerufen werden, indem man auf dem Domino Server die Datenbank „DECS Administrator“ öffnet.



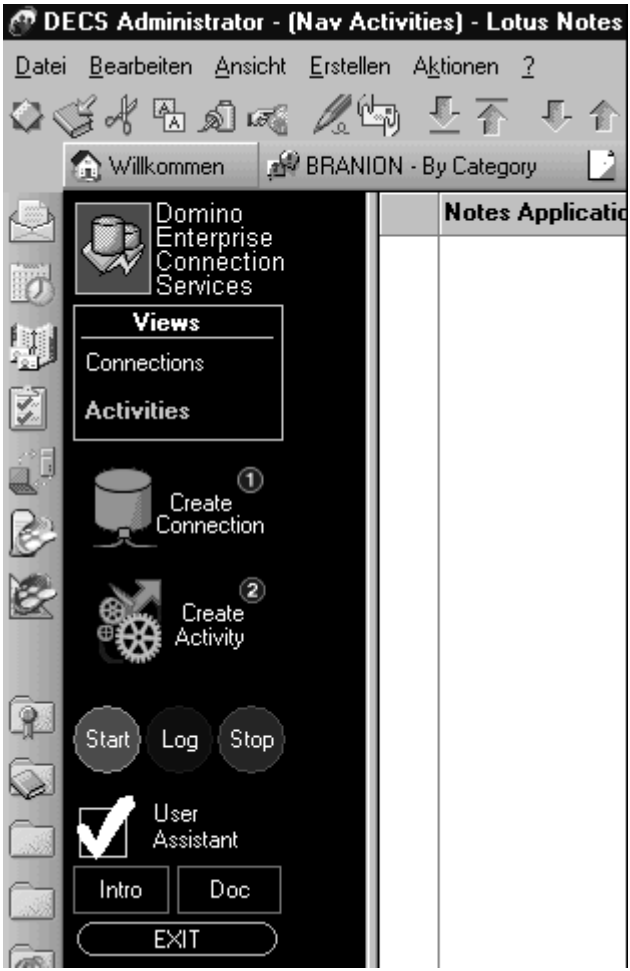
DECS Administrator als Datenbank öffnen

Im Domino Administrator Client kann man den gleichnamigen Eintrag auswählen und damit den DECS Administrator starten.

DECS-Administrator-Eintrag im Administrator Client

Der DECS Administrator zeigt links einen Navigationsbereich zum Aufrufen der verschiedenen Funktionen. Rechts sind die Dokumente zu diesen Funktionen aufgelistet.

Die nachfolgende Abbildung zeigt den Navigationsbereich des DECS Administrators:



Der Navigationsbereich des DECS Administrators

Der DECS Administrator ist eine Notes-Datenbank, die sowohl ein Bearbeitungsprogramm als auch Dokumente enthält. Die Verwendung der Administrator-Datenbank ermöglicht die Servergesteuerte Verarbeitung von externen Daten. Die Definitionen teilen sich auf in Verbindungen zu Datenbanken und durchzuführende Aktivitäten. Dazu kommen Ansichten, in denen die bestehenden Aktivitäten und ihr aktueller Status bzw. Verbindungen angezeigt werden.

Hinter den Icons im Navigationsbereich des DECS Administrators verbergen sich folgende Funktionen:

In der Auswahl "Ansichten" kann zwischen der Anzeige aller definierten Verbindungen und der Anzeige aller definierten Aktivitäten gewechselt werden.

Ansicht auswählen

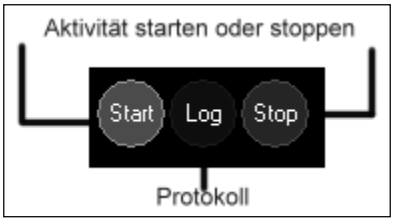
Durch Anwählen dieser Schaltfläche erreicht man die Maske zur Erstellung eines Verbindungsdokumentes.

Verbindung erstellen

Die „Create Activity"-Schaltfläche führt in eine Maske, in der die Definitionen des Real-Time-Zugriffes angegeben werden.

Aktivität erstellen

In Abhängigkeit von der Anwenderunterstützung ändert sich die nach der Auswahl folgende Anzeige:



Aktivität starten oder stoppen

Mit Hilfe der Schaltfläche „Start“ wird die Aktivität gestartet, die in der Anzeige der definierten Aktivitäten ausgewählt wurde. Ist diese Aktivität bereits gestartet, hat die Schaltfläche keine Auswirkung bei erneuter Betätigung. In der Ansicht der Verbindungen hat die Schaltfläche ebenfalls keine Auswirkung!

Analog zu „Start“ kann mit der „Stop“-Schaltfläche eine in Verarbeitung befindliche Aktivität gestoppt werden.

Die „Log“-Schaltfläche gibt Auskunft über den Status der ausgewählten Real-Time-Aktivität. Wird die Aktivität gerade ausgeführt, so werden verschiedene Ablaufinformationen angezeigt. Ist die ausgewählte Aktivität nicht aktiv, so wird ein Protokoll der letzten Aktivitäten angezeigt.

Benutzerunterstützung

Die Schaltfläche „User Assist“ ermöglicht das Ein- und Ausschalten der Benutzerunterstützung. Diese ist sinnvoll für den ungeübten Benutzer der DECS Administration.

Anzeige von Erklärungen und Online-Hilfe

Die Intro-Schaltfläche bewirkt, dass Erklärungen zu den Schaltflächen angezeigt werden.

Die Online-Hilfe wird durch Betätigen der DOC-Schaltfläche aufgerufen.

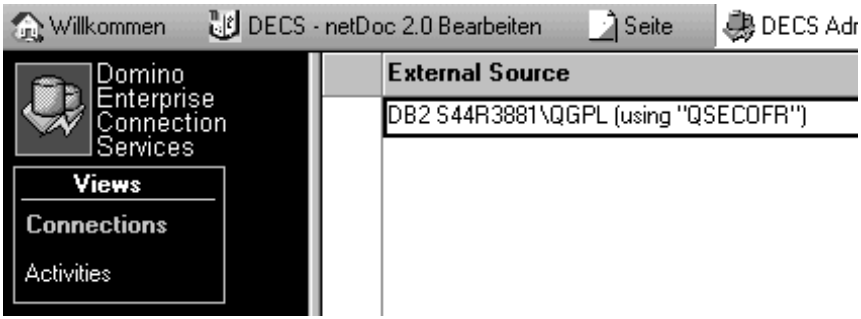
Programm beenden

Mit EXIT wird die DECS-Datenbank verlassen.

8.1.4.1 DECS-Verbindungen

Für den Zugriff auf Nicht-Notes-Daten ist unter DECS eine Verbindung zu definieren. Diese Verbindung (Connection) enthält die Informationen, wie auf externe Daten zugegriffen werden soll und spezifiziert diesen Zugriff.

Die nachfolgende Abbildung zeigt eine Übersicht der definierten Verbindungen. Für jede Verbindung wird ein Eintrag angezeigt:



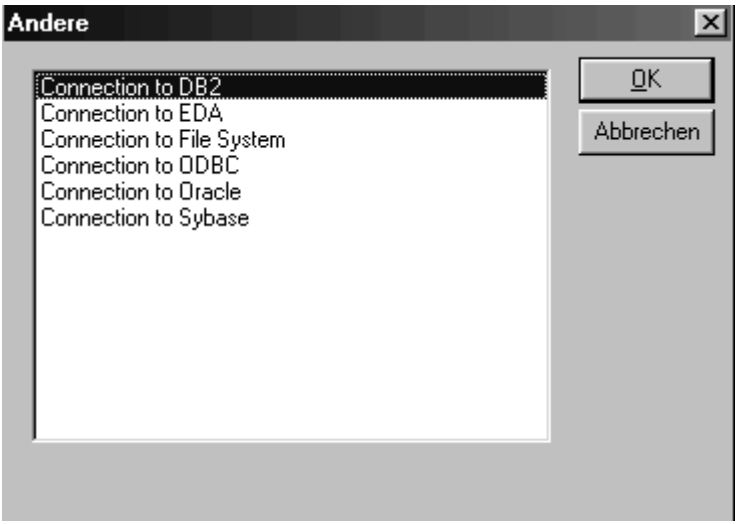
Ansicht Verbindungen

Die Namensvergabe erfolgt nach gewissen Konventionen automatisch während der Erstellung der Verbindung. Bei der Namensvergabe werden

- das externe System
- die Datenbank
- der Dateiname

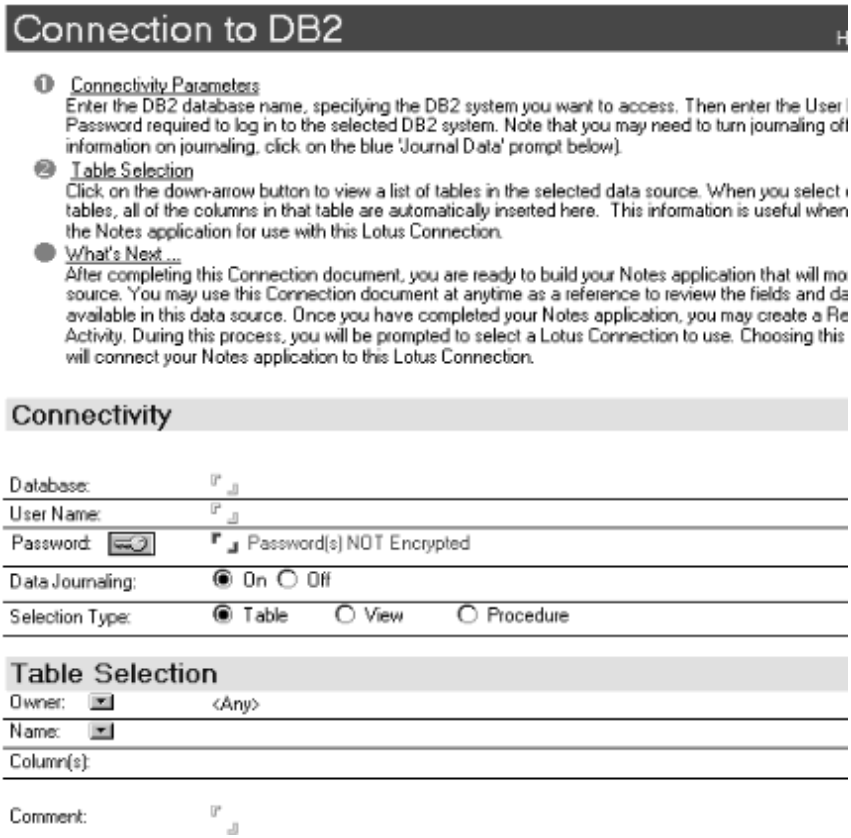
verwendet.

Neue Verbindungen können aus dem DECS Administrator erstellt werden. Hierzu wählt man die Schaltfläche „Create Connection“ aus. Anschließend muss die Art der externen Daten angegeben werden. Ein Auswahlfenster gibt die unterstützten Datenbanksysteme vor:



Auswahl der Datenbanksysteme

Nach Auswahl des Verbindungstyps werden weitere Angaben zu der Verbindung gemacht:



Neue Verbindung definieren

Neben dem Eintrag des Namens der externen Datenquelle erfolgt hier die Anmeldung mit einem entsprechend berechtigten Benutzer auf dieses System.

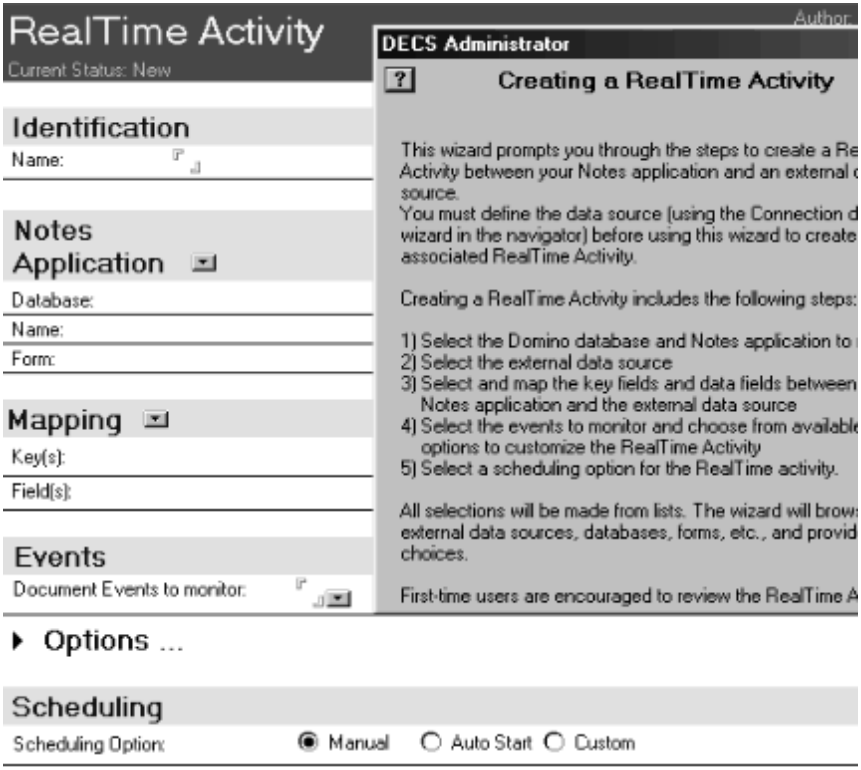
Für das Fortschreiben von Dateien – aus Lotus Domino heraus – kann angegeben werden, ob diese Updates unter Verwendung der Journal-Funktionen erfolgen sollen. Dies ist jedoch nur bei Datenbanken anzugeben, die auch in einem Journal aufgezeichnet werden.

8.1.4.2 DECS-Aktivitäten

In der Maske für Real-Time-Aktivitäten werden die Angaben zu externen Dateien und den entsprechenden Notes-Datenbanken hinterlegt. Voraussetzung für die Definition der Aktivitäten ist eine Verbindung zu einem externen System – das im vorherigen Abschnitt behandelt wurde – und eine vorhandene Notes-Datenbank, die dazu dient, Dateninhalte der externen Datenbank aufzunehmen. Zur Erstellung einer solchen Aktivität wird im DECS Administrator die Schaltfläche „Create Activity“ angeklickt.

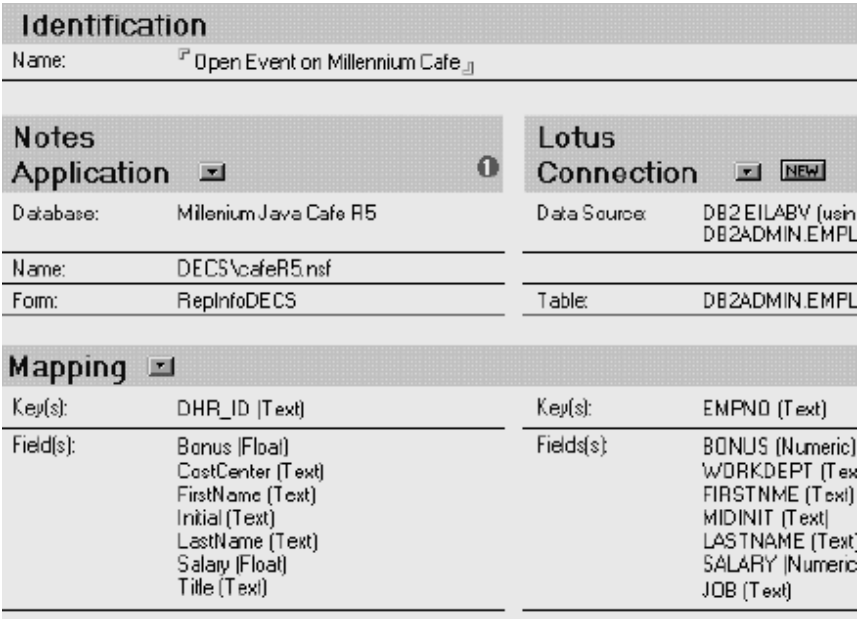
Die Schaltfläche „Create Activity“

Daraufhin erscheint die Bearbeitungsmaske für die Angaben der Real-Time-Aktivitäten:



Aktivität erstellen

Neben dem Bearbeitungsbildschirm wird ein Assistent eingeblendet, der den Definitionsprozess beschreibt. Durch Anklicken von „OK“ wird dieses Fenster geschlossen und es kann mit der Definition der Datenverknüpfungen begonnen werden.



Aktivität erstellen

Der Bildschirm für die Definition der Aktivität ist in zwei Bereiche untergliedert. Im linken Bereich werden die Angaben der Notes-Datenbank eingetragen und im rechten Bildschirmteil die Informationen der externen Datenquelle.

Sind alle notwendigen Angaben erfolgt, so kann die Aktivität gespeichert und der Datenaustausch zwischen Lotus Notes und der externen Datenquelle über DECS gestartet werden.

Der Datenaustausch wird aus dem DECS Administrator heraus gestartet, indem die zu verarbeitende Aktivität aus der Übersicht

ausgewählt und die Schaltfläche “Start” im linken Bereich des Administrators betätigt wird.

Der Datenaustausch kann auch per Programm (also z.B. durch einen Agenten), vom System gesteuert, aufgerufen werden.

8.1.4.3 Metadaten

Um eine Real-Time-Activity (einen Zugriff auf Nicht-Notes-Daten) zu realisieren, müssen noch einige Vorarbeiten geleistet werden. Für jeden Datenzugriff muss mindestens eine Ansicht vorhanden sein. Diese Ansicht stellt das Bindeglied zwischen Notes und den Nicht-Notes-Daten her. In dieser Ansicht definiert man “Metadaten“. Metadaten sind eine Definition der Notes-Felder mit Angabe der externen Felder, deren Daten in den Notes-Feldern dargestellt werden sollen. Es besteht die Möglichkeit, mehrere externe Datenquellen in einer Notes-Ansicht über Metadaten zu integrieren.

Die Metadaten enthalten neben den Feldern, die die Informationen aus den externen Datenquellen aufnehmen sollen, auch die Angabe der Schlüsselfelder. Als Schlüsselfelder können mehrere Felder definiert werden. Es ist aber unbedingt darauf zu achten, dass die Datei, die die zu importierenden Daten enthält, über eindeutige Schlüssel verfügt. Doppelte Schlüssel in einer Datei führen bei der Datenübertragung zu einem Chaos!!

Inhalt einer solchen Metadatei muss mindestens ein Schlüsselfeld und mindestens ein Nicht-Schlüsselfeld sein.

Die Angaben der Metadaten sowie die gesamte Konfiguration des Datenzugriffes auf externe Daten erfolgt in einer Notes-Datenbank. Die Metadaten werden in den Aktivitäten definiert.

8.1.5 DECS und AS/400

DECS ist auch auf der AS/400 seit der Domino-Version 4.6.3 verfügbar. Die AS/400-Version von DECS ermöglicht die Integration von DB2/400-Daten in Notes-Anwendungen. Unter Verwendung von DECS können auf diese Weise AS/400-Daten direkt in Notes-Anwendungen verarbeitet werden.

Damit DECS auf dem System AS/400 einsetzbar ist, müssen neben den speziellen Anforderungen für den Datenzugriff von DECS auch Einstellungen auf der AS/400 vorgenommen werden, um mit Hilfe eines PCs auf die relationale Datenbank des Systems zugreifen zu können. So muss beispielsweise TCP/IP definiert sein, wenn man über dieses Protokoll die Verbindung zwischen PC und AS/400 herstellen will.

Die Berechtigungen sowohl auf der AS/400-Seite als auch auf der Notes-Seite müssen ausreichend sein, um auf die Daten der AS/400 zugreifen zu können (Lese-/Schreibrechte).

8.1.5.1 Installation von DECS auf dem System AS/400

DECS wird automatisch im Zuge der Domino-Installation auf der AS/400 installiert. Die DECS-Funktionalitäten sind in der Option *BASE im LODRUN-Befehl enthalten. Bei der Konfiguration wird der DECS-Prozess angekreuzt.

Die Installation wird auch im Zuge eines Upgrades von einer früheren Domino-Installation automatisch durchgeführt.

Nachdem DECS installiert ist, bedarf es, wenn das Domino Release kleiner als R5 ist, noch einiger Schritte, um DECS zu aktivieren:

- Sicherstellen, dass der Domino Server nicht aktiv ist
- Einstellungen in der NOTES.INI anpassen:
- EXTMGR_ADDINS=decsext.

Existiert bereits ein Eintrag EXTMGR_ADDINS, muss dieser um den Eintrag „decsext“ ergänzt werden.

Um DECS zu starten, können zwei Varianten verwendet werden:

- für einen automatischen Start von DECS

  Beim Starten des Domino Servers in der Datei „NOTES.INI“ die Liste der Server Tasks um folgenden Eintrag erweitern:

  ServersTasks=Replica, Router, Update, Stats,
  AMgr,. .,DECS

- um DECS manuell zu starten

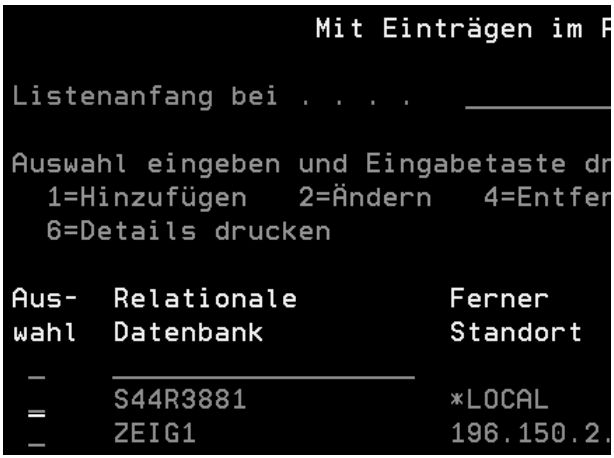
  An der Domino-Konsole „LOAD DECS“ eingeben.

Damit ist der DECS-Server startbereit.

8.1.5.2 Vorbereitungen auf dem System AS/400

Für einen Datenzugriff auf eine DB2/400-Datei ist die Berechtigung für das Lesen und Schreiben von Datensätzen notwendig. Diese Berechtigung kann, falls nicht vorhanden, entweder direkt auf der AS/400 eingegeben werden (EDTOBJAUT) oder aber unter Verwendung des Operationsnavigators in grafischer Form editiert werden.

Weiterhin ist für den Zugriff auf die DB2/400 ein Eintrag im RDB-Verzeichnis der AS/400 erforderlich. Dieser kann mit dem Befehl WRKRDBDIRE gepflegt und geprüft werden. Es ist darauf zu achten, dass für die relationale Datenbank der AS/400 ein Eintrag *LOCAL als ferner Standort angegeben ist. Sollte dies nicht der Fall sein, ist ein solcher Eintrag vorzunehmen, da über diese Definition die Verknüpfung zwischen Lotus Notes und der DB2/400 vorgenommen wird.



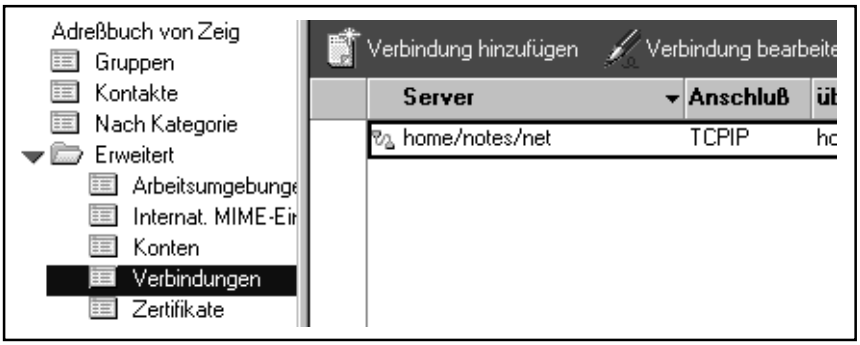
Der Befehl WRKRDBDIRE

8.1.5.3 Das Verbindungsdokument für DECS

Die Verbindung zwischen Lotus Notes und der DB2/400 einer AS/400 setzt neben den zuvor beschriebenen Anforderungen auf der AS/400-Seite auch noch einige Grundeinstellungen in Notes voraus. Hier ist es nämlich erforderlich, Notes erkenntlich zu machen, wie und auf welche DB2/400 zugegriffen werden soll. Für solche Einstellungen arbeitet Lotus Notes mit sogenannten Verbindungsdokumenten. Das sind Eintragungen im öffentlichen Namens- und Adressbuch (dem Domino-Verzeichnis). Sie definieren, wie und über welche Anschlüsse auf externe Server zugegriffen werden soll. Das folgende Beispiel zeigt, wie man eine Verbindung zu einer AS/400 definiert:

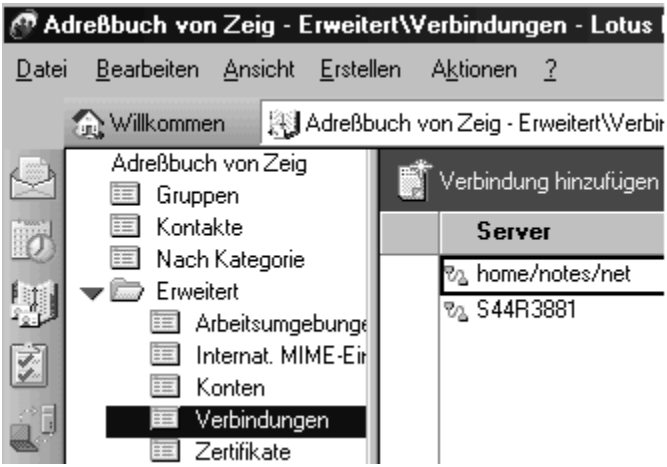
Die Konfiguration erfolgt in einem Notes Client. Es ist darauf zu achten, dass der Notes User über ausreichende Rechte verfügt, um das Adressbuch von Notes (nicht das persönliche Adressbuch des Notes-Users!) zu bearbeiten.

Im Domino-Verzeichnis befindet sich hinter der Schaltfläche „Erweitert“ eine Untergruppe mit der Bezeichnung „Verbindung“.



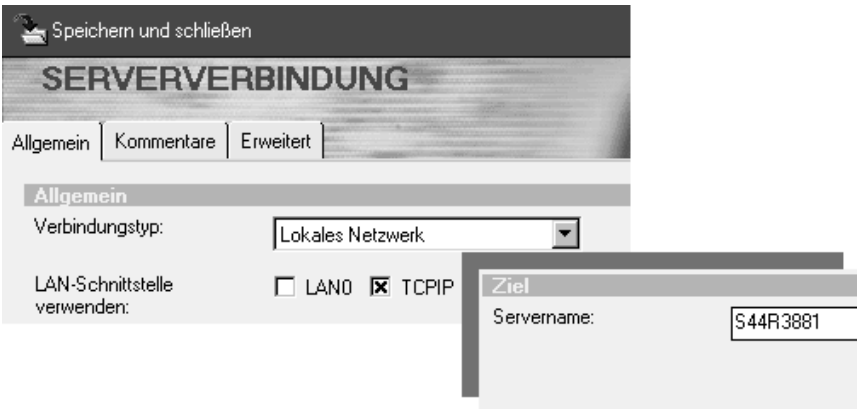
Verbindungen

Sollte in dieser Anzeige kein Eintrag für die AS/400 vorhanden sein, so ist noch keine Verbindung zwischen Lotus Notes und der DB2/400 definiert. Befindet sich bereits ein Eintrag in der Übersicht, so müssen die weiteren gezeigten Konfigurationsschritte nicht ausgeführt werden.



Verbindung hinzufügen

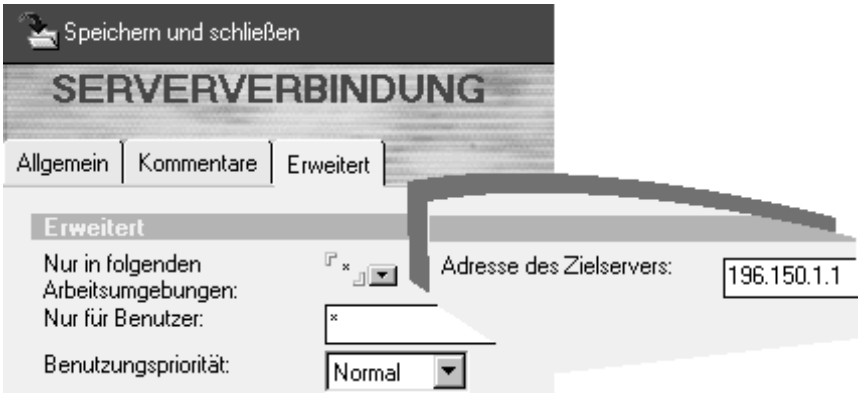
Durch Auswahl der Schaltfläche „Verbindung hinzufügen“ wird die Definitionsmaske geöffnet:



Allgemeine Angaben zur Verbindung

Als Verbindungstyp ist „Lokales Netzwerk“ anzugeben. Im Feld „Servername“ ist der Name des AS/400-Systems einzutragen (ist der nicht bekannt, kann dieser unter Verwendung des Befehls DSPNETA angezeigt werden). In der Auswahl „LAN-Schnittstelle verwenden“ ist das Kästchen „TCP/IP“ anzuklicken.

Anschließend müssen in dem Bereich „Erweitert“ noch Eintragungen vorgenommen werden:



Serververbindung angeben

Soll die Verbindung nur in bestimmten Arbeitsumgebungen genutzt werden, so sind diese unter „nur in folgenden Arbeitsumgebungen“ einzutragen. Ein * bedeutet, dass die Einstellungen dieser Verbindung in allen Arbeitsumgebungen verwendet werden können.

Unter „Adresse des Zielservers“ erfolgt die Eintragung der TPC/IP-Adresse der AS/400.

Mit der Schaltfläche „Speichern und Schließen“ wird die neu erstellte Verbindung gespeichert und erscheint nun in der Übersicht der konfigurierten Verbindungen.

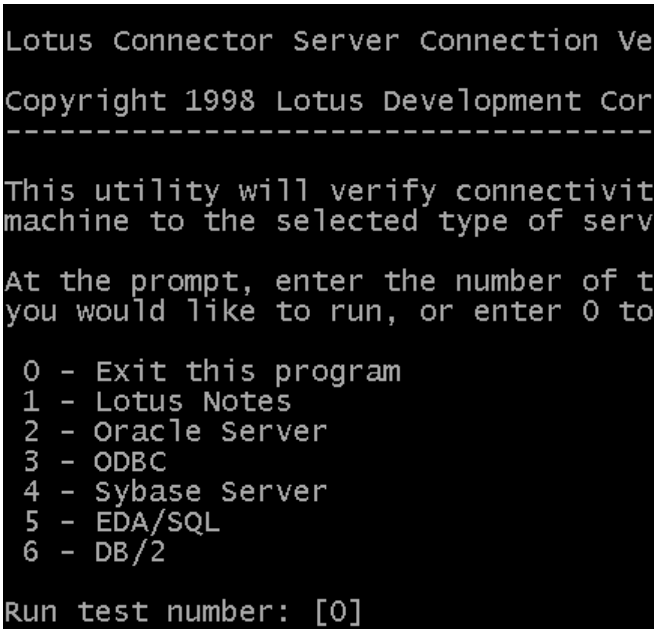
8.1.5.4 DECS-Verbindung testen

Um über eine solche Verbindung mittels DECS erfolgreich auf Daten einer AS/400 zugreifen zu können, muss die Verbindung zu der AS/400 (genauer zur DB2/400) korrekt eingerichtet sein. Offensichtlich liefert Lotus – wegen immer wiederkehrender Probleme und Fragen mit DECS – ein Testtool aus, mit dem man in der Lage ist, die Verbindung zu einem externen Datenbanksystem zu überprüfen. Dieses Tool (im Falle einer Verbindung von WIN95/NT) – nlctest – befindet sich sowohl im Hauptverzeichnis des Domino Servers als auch im Hauptverzeichnis des Notes Clients, wenn auch ein Administrator Client installiert wurde. Je nach Client hat es unterschiedliche Namen:

- nlctest für Windows NT/95/98
- lctest für AIX
  1. ilctest für OS/2
  2. alctest für Windows Alpha NT
- lctest für UNIX

Das nachfolgende Beispiel zeigt den Verbindungstest zwischen einem Lotus Notes Client auf einem Windows-NT/95/98-Rechner und einer DB2/400 auf einer AS/400.

Nach Aufruf des Programms nlctest.exe erscheint die folgende Anzeige:



lctest

Nach Auswahl 6 (DB/2) versucht das Tool einen Datenzugriff auf die in der Folgeanzeige zu spezifizierende Datenbank. Hierbei muss neben dem Namen des zu verbindenden Datenbanksystems der Benutzer und das Benutzerkennwort angegeben werden. Nach der Eingabe dieser Parameter wird der Test gestartet. Im Fehlerfall wird die Fehlermeldung – mit einem Fehlercode spezifiziert – ausgegeben. Endet der Test erfolgreich, so erscheint eine entsprechende Bestätigungsmeldung.

Diese Seite ist zur Zeit nicht belegt.

8.1.6 DECS starten

In der Regel startet DECS automatisch beim Hochfahren des Domino Servers. Um dies zu kontrollieren, geht man in die Anzeige des Domino Servers. Der Eintrag „DECS Server started“ zeigt an, dass der DECS-Server-Prozess gestartet wurde.

```
2000 09:20:10   DECS Server started
```

Server-Meldung „DECS gestartet“

Das automatische Starten der DECS Server Tasks wird in der Datei „NOTES.INI“ gesteuert. Der erforderliche Eintrag für das automatische Starten des DECS-Server-Prozesses ist in dem Eintrag „ServerTasks“ zu definieren. Sollte eine Version kleiner als 5.0.2 von Domino eingesetzt werden, ist es unter Umständen erforderlich, den Eintrag für das automatische Starten dort einzutragen. Die nachfolgende Abbildung zeigt ein Beispiel für solch einen Eintrag:

```
Datei  Bearbeiten  Suchen  Fenster  Optionen  Hilfe
                C:\lotus\domino\notes.ini
NAMEDSTYLE2_FACE=Default Sans Serif
DefaultMailTemplate=mail50.ntf
ServerTasks=Event,Sched,Stats,HTTP,DIIOP,NNTP,DECS,maps
ServerTasksAt1=Catalog,Design
```

NOTES.INI – DECS-Eintrag

Anmerkung:

Die NOTES.INI findet man im Normalfall im Hauptverzeichnis des Domino Servers. Sie kann mit einem beliebigen Textverarbeitungsprogramm editiert werden. Für die DECS Funktionalitäten muss „DECS“ im Eintrag „ServerTasks“ vorhanden sein. Ist kein automatischer Start von DECS gewünscht, so kann DECS über den Domino Administrator einzeln gestartet werden.

8.2 Lotus Enterprise Integrator LEI

Eine der Stärken von Domino ist das Verteilen von Informationen über verschiedene Server hinweg bis zum dezentralen Anwender, der sich nur hin und wieder in ein Netzwerk einklinkt. Wenn eine Domino-Anwendung mit den Daten einer zentralen Datenbank wie z.B. einer AS/400 arbeitet, dann reichen die normalen Datenzugriffsmechanismen wie etwa ODBC nicht mehr aus. Der dezentrale Anwender müsste sonst jedesmal eine Verbindung zum System aufbauen. Eine Möglichkeit, die ständige Aktualisierung solcher Daten zu gewährleisten, stellt der Einsatz von LEI (früher: NotesPump) dar, einer Software zum regelmäßigen Datenabgleich und -austausch zwischen Lotus Domino und verschiedenen Datenbanken.

Diese Seite ist zur Zeit nicht belegt.

8.2.1 Was ist LEI?

LEI ist entweder eine Erweiterung zum Domino Server oder ein eigener Server. Es hat die Aufgabe zeit- oder Event-gesteuert einen Dateitransfer zwischen einer Quell- und einer Zieldatenbank durchzuführen. Damit integriert LEI relationale Datenbanken, ERP-Systeme (ERP = Enterprise Resource Planning) sowie text- und transaktionsorientierte Systeme in die Datenbankwelt von Notes und Domino. In der Regel ist eine der verbundenen Datenbanken eine Notes-Datenbank. Das ist aber nicht zwingend so.

Wie auch DECS unterstützt LEI die Real-Time-Integration von Datenbanken. Es bietet zusätzlich:

- Zeitgesteuerte Datenabgleiche und Datenübernahmen im Hintergrundbetrieb
- einmalige Datenübernahmen
- Überarbeitung der Daten durch Agenten
- Archivierung von Datenbanken

LEI besteht aus drei Komponenten:

- dem LEI Server
- dem LEI Administrator
- dem LEI-Entwicklungs-Client

Damit LEI seine Arbeit aufnehmen kann, muss bereits ein Domino Server gestartet sein. Der LEI Administrator und der Entwicklungs-Client sind in einer Domino-Datenbank enthalten, die über einen normalen Notes Client gestartet wird.

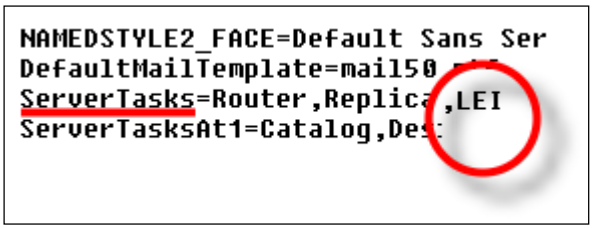
Diese Seite ist zur Zeit nicht belegt.

8.2.2 Die Installation

Die Installation von LEI erfolgt aus dem zur Hardware passendem Unterverzeichnis der CD. Dementsprechend hängt auch der Aufruf der Installationssoftware von der Hardwareplattform ab. Da LEI ohne einen Notes Client nicht administriert werden kann, muss dieses Programm vorher installiert werden.

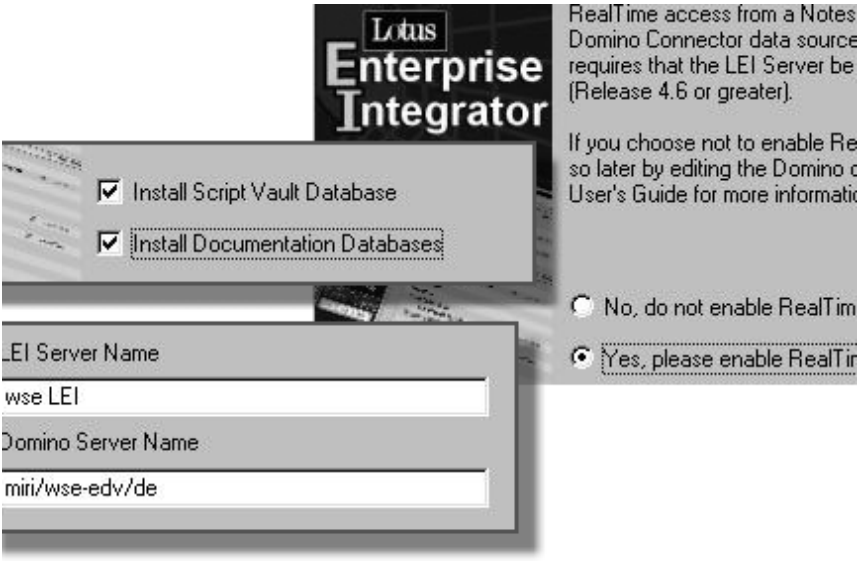
8.2.2.1 Die Installation unter Windows

Für die Installation unter Windows wird das Programm „Setup.exe“ aus dem Verzeichnis „Win32“ gestartet. Es folgt eine Reihe von Dialogen, die alle zur Installation erforderlichen Informationen erfragen. Es ist zu entscheiden, ob LEI als eigenständiger Server oder als Prozess eines Domino Servers arbeiten soll.



LEI und der Domino Server

Davon abhängig sind weitere Angaben zu Server und Administrator zu machen. Wichtig ist sicher auch, ob die Real-Time-Bestandteile aktiviert werden sollen.



Weitere Angaben zur Installation

Nach der Installation zeigt LEI eine Meldung in der Server-Konsole, dass die Initialisierung erfolgreich durchgeführt wurde.

```
Initializing Lotus Enterprise Integrator (Build 091119)
14/06/2000 09:08:23 PM  LEI: Initialization successful
```

LEI-Server-Konsol-Meldung

8.2.2.2 Die Installation auf dem System AS/400

Da LEI auf einen Domino Server aufsetzt, müssen alle Hard- und Softwarevoraussetzungen des Domino Servers erfüllt sein: Risc-Prozessor, OS/400 V4R2 und zusätzlich die Option 6 HiTest C API (siehe Domino-Installation). Mit Hilfe des Befehls DSPSFWRSC können die installierten Lizenzprogramme überprüft werden. Die eigentliche Installation wird mit Hilfe des LODRUN-Befehls durchgeführt, nachdem die CD mit dem LEI-Lizenzprogramm in das CD-Laufwerk eingelegt wurde:

LODRUN DEV(*OPT) DIR(‚/OS400‘)

Mit Hilfe des DSPSFWRSC-Befehls kann die erfolgreiche Installation überprüft werden.

Jetzt wird der LEI Server dem Domino Server bekannt gemacht. Dazu muss der Domino Server gestartet sein und der Befehl ADDLEISVR gestartet werden (der Benutzer, der diesen Befehl startet, muss folgende Rechte haben: *ALLOBJ, *SECADM, *IOSYSCFG und *JobCTL).

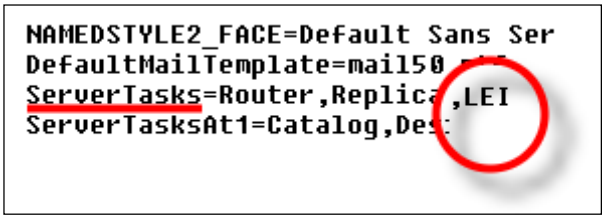
Der Befehl hat folgende Parameter:

- SERVER
  Name des LEI Servers
- LCLLEISVR
  Name des Domino Servers
- DOMSVRDB
  Name des Domino Servers, zu dem der Administrator für den LEI Server gehört.

- CRTDB
  Administrator-Datenbank anlegen (*YES, *NO)
- AUTOADD
  Soll der LEI Server automatisch gestartet werden (*Yes, *NO)
- ADMDB
  Name der Administrator-Datenbank (Standard: LEIADM.NSF)
- DBMGR
  Name des Datenbankmanagers

Damit ist die Installation abgeschlossen.

Wird bei der Installation der Parameter AUTOADD mit *YES bestätigt, ändert das Installationsprogramm die Datei „NOTES.INI“ ab. Der Eintrag „ServerTasks“ wird um den LEI Server ergänzt. Damit wird LEI von dem Domino Server automatisch gestartet.



LEI-Eintrag in der Datei „NOTES.INI“

Das manuelleStarten von LEI erfolgt über eine Domino-Konsole (WRKDOMCSL) mithilfe des Befehls: LOAD LEI. Beendet werden kann der LEI Server aus der Konsole mit: TELL LEI QUIT. Ob der Server gestartet ist, kann mit WRKACTJOB nachgesehen werden. Die Jobs, beginnend mit „LEI“, laufen im Subsystem des entsprechenden Domino Servers.

8.2.3 Der Administrator

Wer mit dem DECS Administrator gearbeitet hat, wird sofort die Verwandtschaft der beiden Tools erkennen. Das Aussehen des LEI Administrators und seine Bedienungselemente sind offensichtlich das Vorbild für DECS gewesen.

Der LEI Administrationsnavigator

Im Administrationsbereich finden sich Icons mit den folgenden Funktionen:

In der Auswahl “Ansichten” kann zwischen der Anzeige aller definierten Verbindungen und der aller definierten Aktivitäten gewechselt werden. Dazu kommt eine Gesamtübersicht und eine Ansicht nach Kategorien.

Ansicht auswählen

Durch Anwählen der Schaltfläche „Create Connection“ erreicht man die Maske zur Erstellung eines Verbindungsdokumentes.

Verbindung erstellen

Die „Create Activity“-Schaltfläche führt in eine Maske, in der die Definitionen für eine der möglichen Aktivitäten definiert werden.

Aktivität erstellen

Mit Hilfe der Schaltfläche „Start“ wird die Aktivität gestartet, die in der Anzeige der definierten Aktivitäten ausgewählt wurde. Ist diese Aktivität bereits gestartet, hat die Schaltfläche keine Auswirkung bei erneuter Betätigung.

Aktivität starten oder stoppen

Analog zu „Start“ kann mit der „Stop“-Schaltfläche eine in Verarbeitung befindliche Aktivität gestoppt werden.

Anzeige von Protokoll, Online-Hilfe und Erklärungen

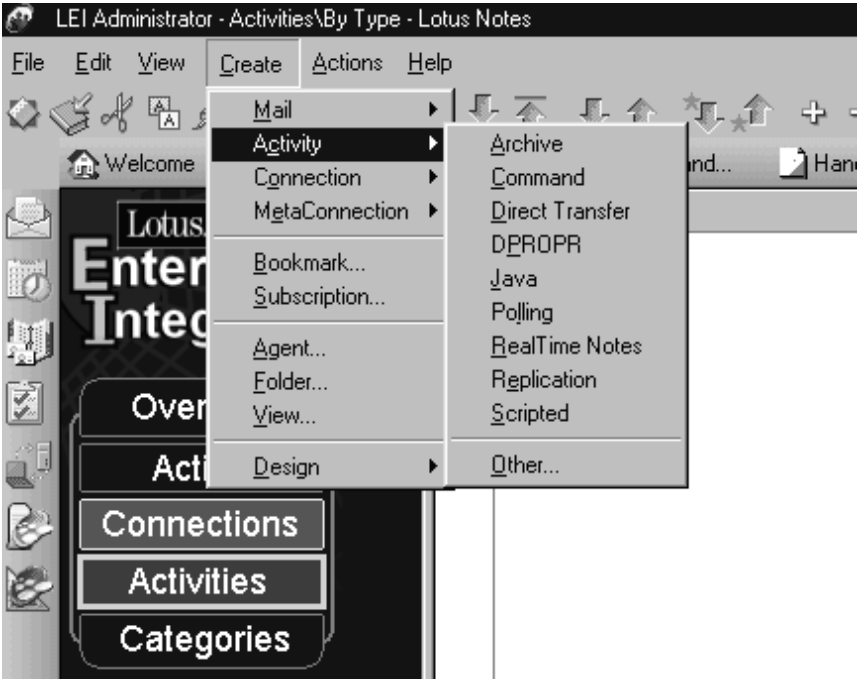
Über „Log“ wird zur Protokoll-Datenbank verzweigt.

Das Betätigen der „Intro“-Schaltfläche bewirkt, dass Erklärungen zu den Schaltflächen angezeigt werden.

Die Online-Hilfe wird durch Anklicken der „Help“-Schaltfläche aufgerufen.

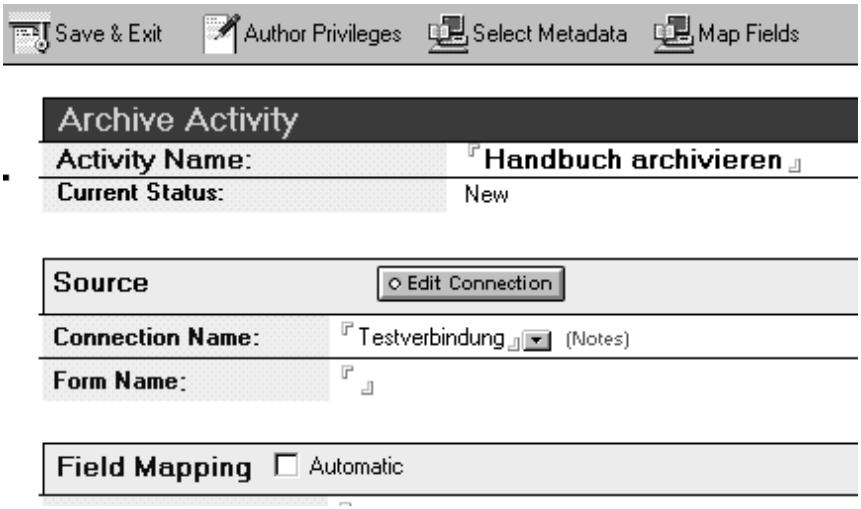
8.2.3.1 LEI-Dokumente bearbeiten

LEI-Dokumente werden aus dem „Create“-Menü des Administrators heraus erstellt.



LEI-Dokument erstellen

Dabei werden innerhalb der unterschiedlichen Dokumentenarten jeweils die möglichen Varianten angezeigt. Die Dokumente haben einen standardisierten Aufbau.



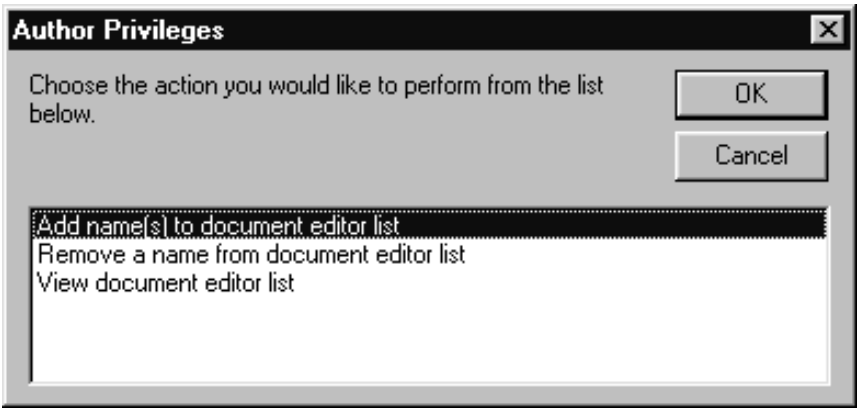
LEI-Dokument

In der Überschrift wird der Dokumententyp angezeigt. Jedes Dokument enthält die Felder “Kategorien” und “Kommentar”. In der Aktionsleiste befinden sich alle oder einige der folgenden Schaltflächen:

Schaltfläche Author Privileges

Mit Hilfe der Schaltfläche „Author Privileges“ wird definiert, wer das Dokument bearbeiten darf.

Dazu wird im folgenden Dialogmenü eine Zeile ausgewählt:



Dialog zu den Author Privileges

Danach können dann ein Benutzer aus dem Domino-Verzeichnis ausgewählt, ein Benutzer aus dem Dokument entfernt oder nur alle berechtigten Benutzer angezeigt werden.

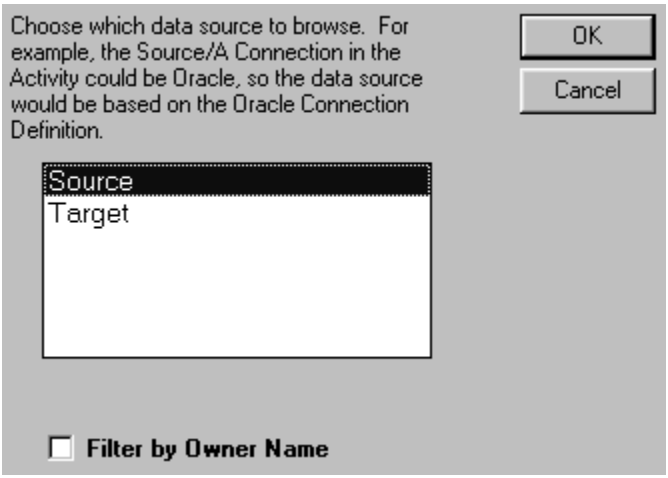
Die Schaltfläche „Run ASAP!“ bewirkt eine Ausführung einer Aktivität sobald der LEI Server dazu in der Lage ist.

Schaltfläche Run ASAP!

Das heißt, dass der Server die Aktivität sofort ausführt, wenn er das nächste Mal die Administrator-Datenbank durcharbeitet.

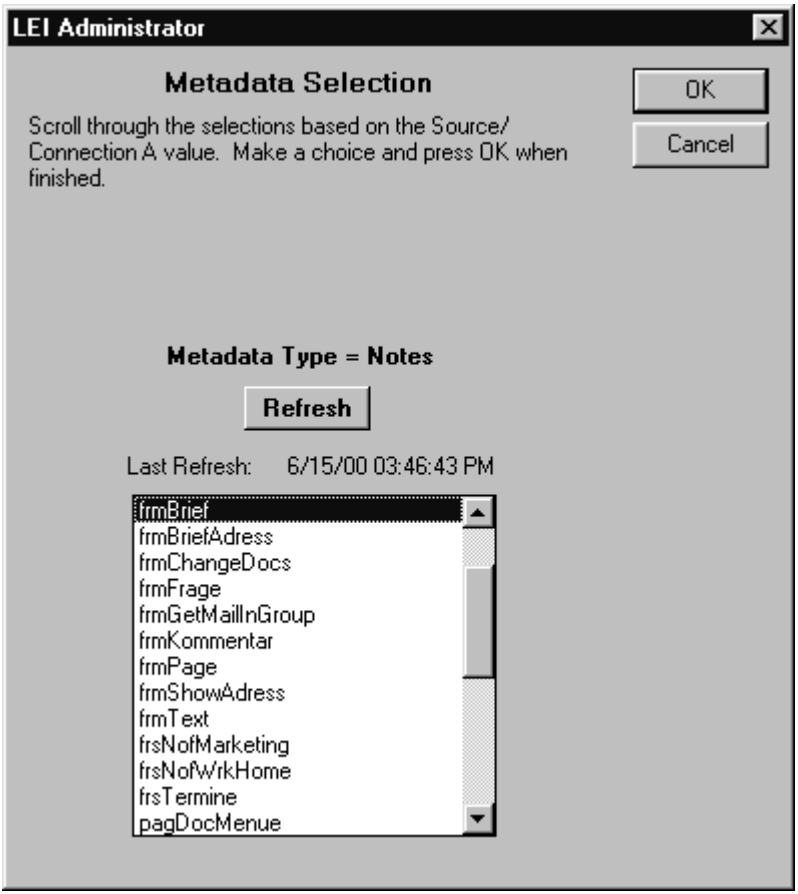
Schaltfläche Metadaten

Je nach Aktivität kann sowohl in Richtung auf die Datenquelle als auch in Richtung auf das Datenziel eine Definition von Tabellen zu Metadaten erfolgen. Dazu wird zunächst die Richtung ermittelt:



Abfrage zur Datenquelle

Im nächsten Schritt werden die verfügbaren Tabellen der ausgewählten Datenquelle zur Auswahl angezeigt.

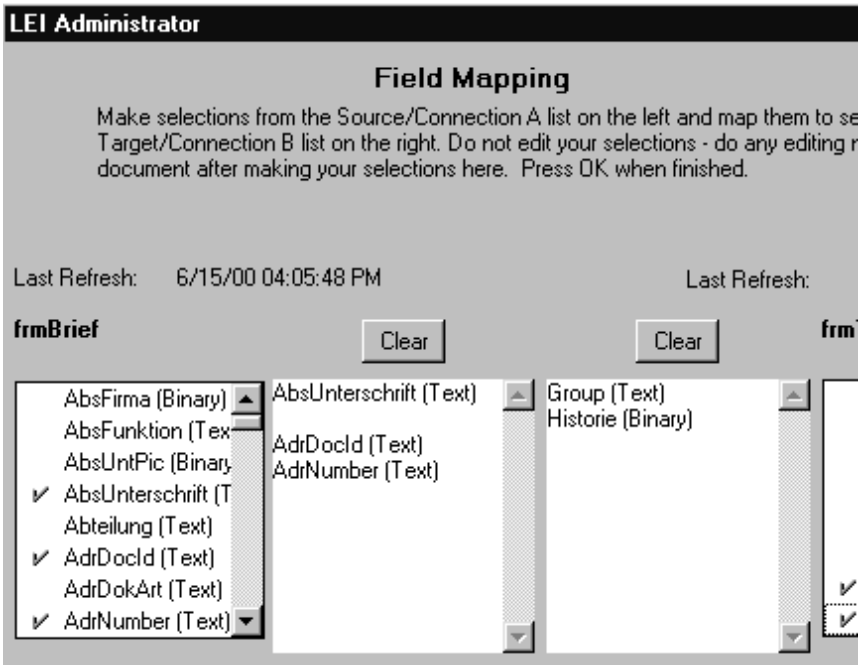


Metadaten auswählen

Das ausgewählte Ergebnis wird in die Aktivität übernommen.

Felder auswählen

Zur Beschränkung der Datenübertragung ist es möglich, die Felder in der Datenquelle zu beschränken und für das Ziel entsprechende Felder anzugeben.



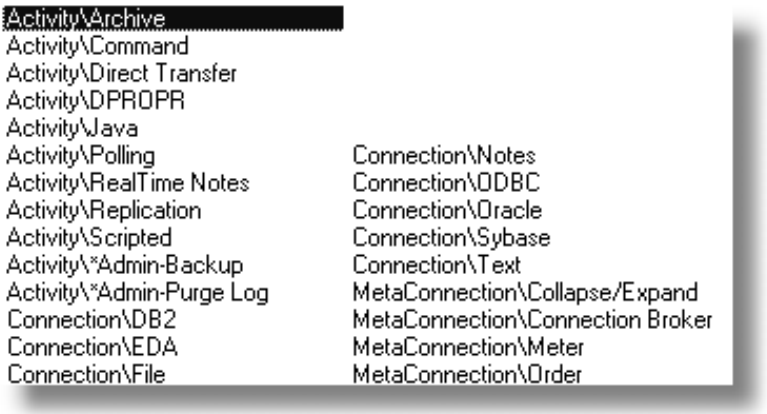
Feldauswahldialog

Wird automatische Feldwahl angekreuzt, so entfällt der Feldauswahldialog.

Speichern und Verlassen

Weitere Schaltflächen erscheinen je nach Dokumentenart zu speziellen Zwecken (z.B. die „Create Agent“-Schaltfläche).

Folgende Dokumentenarten sind verfügbar:



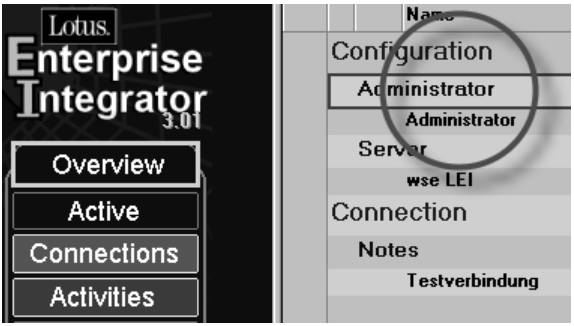
Dokumentenarten

Sie gruppieren sich in:

- Verbindungen
- Aktivitäten
- Metadaten
- Konfiguration

8.2.3.2 Administrator-Konfiguration

In der Administrator-Konfiguration werden Angaben zur Administrator-Datenbank hinterlegt, die für alle LEI Server gelten, die diese Datenbank verwenden.



Die Ansicht „Overview“

Es handelt sich dabei um wenige Standardangaben zu den verwendeten Datenbanken und ihrer Verarbeitung:

LEI Administrator Configuration

Databases located on **miri/wse-edv/de** Domino Server

Log Database:	leilog.nsf
LEI Help Database:	leidoc.nsf
Scripted Agent Database:	leivlt.nsf
Status Broadcast Interval:	60 minutes
Category(s):	
Comments:	

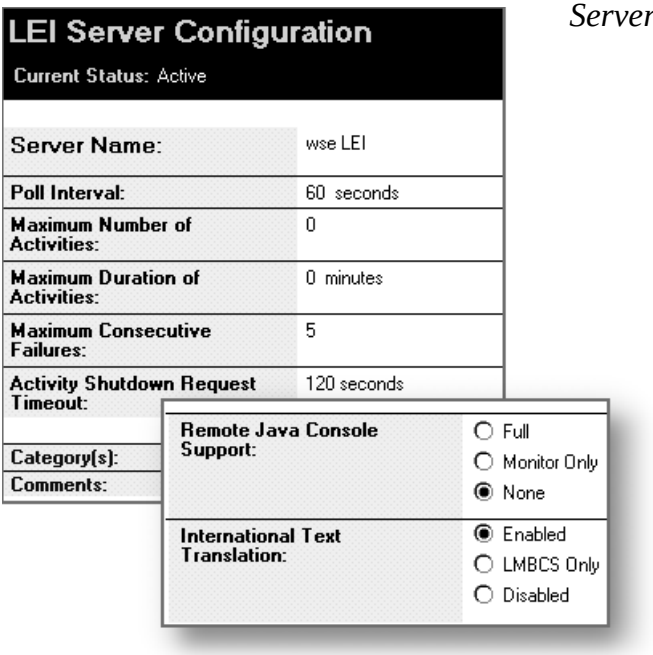
Das Konfigurationsdokument für den Administrator

Das Status-Überprüfungs-Intervall gilt für die Server und sagt aus, in welcher Zeitspanne die Administrator-Datenbank verarbeitet werden muss.

8.2.3.3 Server-Konfiguration

Für jeden Server, der die Administrator-Datenbank verwenden soll, muss ein Server-Dokument mit der entsprechenden Konfiguration angelegt werden.

Server-Konfiguration



In diesem Dokument werden neben dem Verarbeitungsintervall verschiedene Maximalwerte festgelegt, um die Performance des Servers zu steuern. In der Kopfzeile werden die Werte für Status, letzte Verarbeitung und der Autor angezeigt.

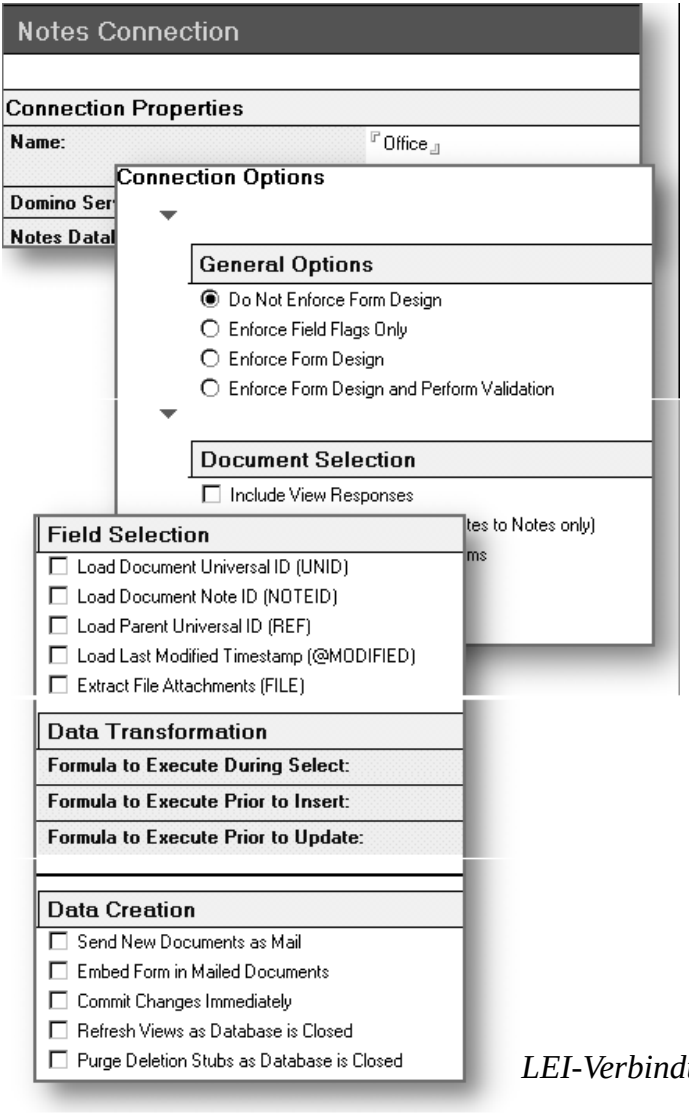
8.2.3.4 Verbindungen

LEI unterstützt sowohl Verbindungen zu Standarddatenbanken als auch solche zu ERP-Systemen wie SAP, PeopleSoft u.a. Es nutzt dabei sogenannte Konnektoren und Programm-Module, die Lotus für die verschiedenen Systeme anbietet. Um die Konnektoren zu speziellen Datenbanken verwenden zu können, müssen sie deshalb getrennt installiert sein.

Auf folgende Datenbanken und ERP-Systeme ist zur Zeit ein Zugriff möglich:

- DB2 in den verschiedenen Versionen
- Notes
- Dateisystem
- EDA/SQL
- Sybase
- Oracle
- ODBC
- JDBC
- ZMerge Text
- JD Edwards
- PeopleSoft
- SAP
- Oracle Applications

Für die Entwicklung von Konnektoren stellt Lotus ein Toolkit zur Verfügung. Es ist damit zu rechnen, dass weitere Datenquellen verfügbar werden.



LEI-Verbindungen

Die Verwendung der angezeigten Felder hängt von der Aktivität ab, für die die Verbindung eingesetzt werden soll.

Verbindungen testen mit lctest.exe

Ob eine Verbindung einsetzbar ist oder nicht, kann mit Hilfe der Programme LCTEST und CONTEST überprüft werden. Die echten Programmnamen sind dabei vom Betriebssystem abhängig (z.B.ICONTEST.EXE für OS/2).

8.2.3.5 Aktivitäten

Mit Hilfe der Aktivitäten wird die eigentliche Übertragungsarbeit definiert. LEI kennt folgende Aktivitäten:

- Command

  Solche Aktivitäten können aus einer Befehlszeile des Betriebssystems heraus gestartet werden. Der Aufruf lautet:

  Xleiact „Name der Aktivität“

  Dabei steht X für einen Kennbuchstaben des jeweiligen Betriebssystems. Unter Unix lautet der Name des Programmes „leiact.exe“, unter Windows „nleiact.exe“. Unter OS/400 wird es mit folgendem Aufruf gestartet:

  call qnoteslei/leiact

- Admin Backup

  erzeugt ein Backup der Administrator-Datenbank auf einem anderen Server.

- Direkte Übertragung

  steuert eine Datenübertragung von einer Datenbank zu einer anderen. Die Verbindung zwischen den Datenbanken muss bereits vorhanden sein, wenn die Aktivität definiert wird. Das gleiche gilt auch für die Datenbanken selbst. Die Feldauswahl kann automatisch oder über eine Feldliste erfolgen. Über eine SELECT-Anweisung kann eine Satzauswahl vorgenommen werden.

- Polling

  Die Polling-Aktivität entspricht inhaltlich einer direkten Übertragungsaktivität. Zusätzlich wird in regelmäßigen Abständen

geprüft, ob eine bestimmte Bedingung erfüllt ist. Nur wenn das der Fall ist, wird die Übertragung durchgeführt. Diese Aktivität kann z.B. dazu genutzt werden, Daten zu übertragen, wenn ein Datenfeld einen bestimmten Wert erhält: alle neuen und geänderten Adressen einer bestimmten Postleitzahl werden übertragen.

- Real-Time

  Diese Aktivität entspricht dem DECS in Domino. Aus einer Notes-Anwendung heraus kann ein direkter Zugriff auf externe Daten erfolgen. Die Steuerung des Zugriffs erfolgt in diesem Fall über den LEI-Server. Diese Form des Zugriffs umgeht die Speicherung redundanter Daten in Lotus-Domino-Datenbanken. Damit ist diese Methode für mobile Clients nicht verwendbar.

- Replikation

  Die Replikationsaktivität synchronisiert die Daten zweier Datenbanken (bzw. Ausschnitte davon über Metadaten). Dieser Vorgang kann einmalig oder z. B. über eine Polling-Aktivität in regelmäßigen Abständen ausgeführt werden. Diese Form des Datenabgleichs bietet sich dann an, wenn die Daten nicht unbedingt aktuell sein müssen, aber mobile Clients mit der Domino-Datenbank in bestimmten Intervallen replizieren.

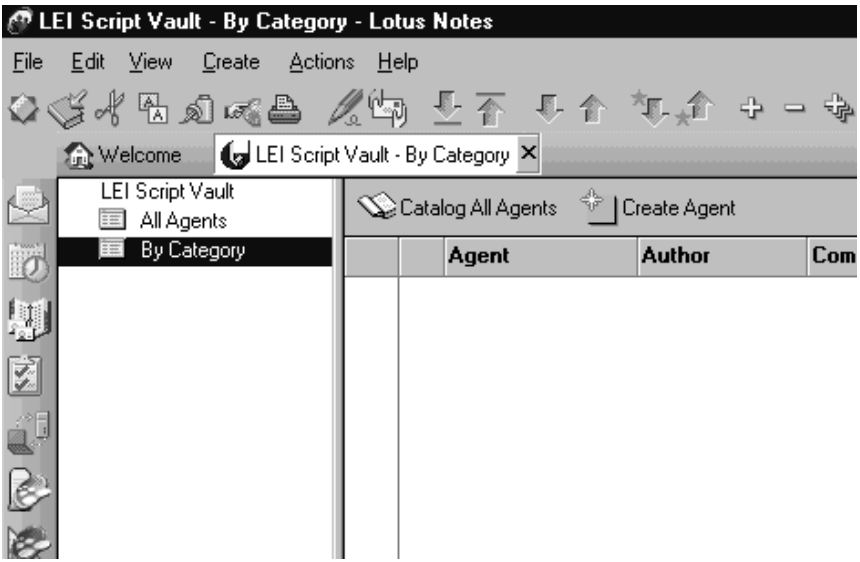
- Script gesteuert

  LEI verfügt über ein eigenes LotusScript-Programminterface. Die LEI-Erweiterungen zu LotusScript erfordern den Eintrag:

  uselsx „Name der lsx-Datei“

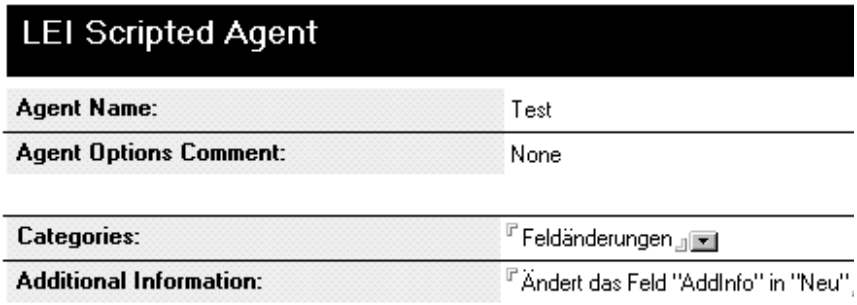
  im Options-Ereignis des Agenten. Der Agent kann in einer beliebigen Domino-Datenbank gespeichert sein. Die Standard-

datenbank von LEI für die Aufnahme von Agenten heißt „LEI Script Vault“ (leivlt.nsf) und wird automatisch installiert.



Die Datenbank „LEI Script Vault“

Über die Schaltfläche „Catalog All Agents“ werden Dokumente für die Agenten erstellt, die in dieser Datenbank enthalten sind.



Agenten-Beschreibungsdokument

Wird ein Agent aus der Datenbank gelöscht, so wird über die Schaltfläche auch das beschreibende Dokument aus der Datenbank gelöscht.

Über die Verwendung von Verbindungen kombiniert mit einer oder mehreren Aktivitäten lassen sich sehr viele Datenübertragungsanforderungen erfüllen. Schon nach wenigen Einsätzen stellt LEI oft die wirtschaftlichere Lösung als eine eigene Programmschnittstelle dar. Leider fehlen einige Funktionen (z.B. die Erstellung eines Feldes im Wege einer Rechenoperation aus Feldern der Datenquelle), sodass die eigene Programmierung nicht immer erspart bleibt. Aber auch in diesem Fall reduziert sich der Aufwand dann auf die Nachbearbeitung der übernommenen Daten mit Hilfe eines Agenten.

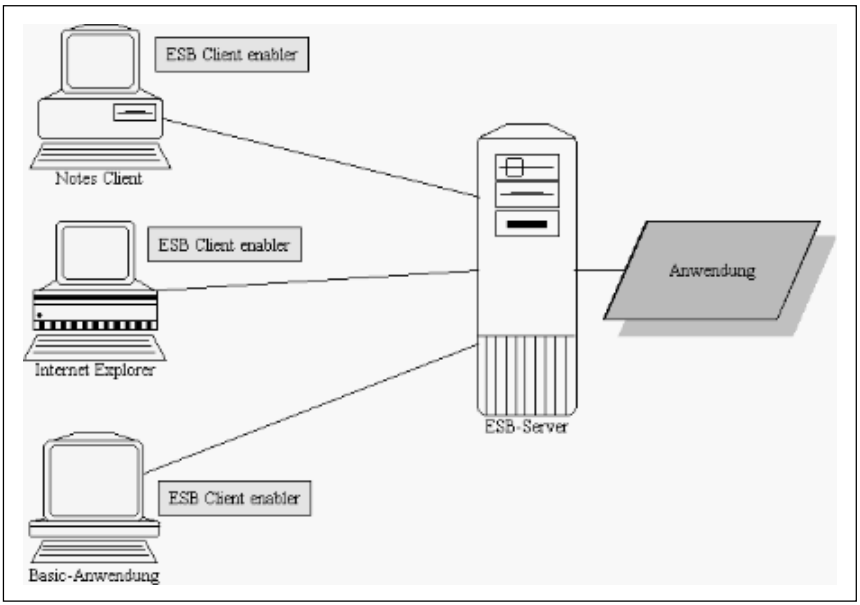
Diese Seite ist zur Zeit nicht belegt.

8.3 Der Enterprise Solution Builder (ESB 3.02)

Der ESB ist noch nicht lange verfügbar und wird in der Öffentlichkeit noch nicht weiter herausgestellt. Er wird jetzt von Lotus ausgeliefert und kann getestet werden.

Was ist nun eigentlich unter dem Enterprise Solution Builder zu verstehen? Lotus hat dieses Produkt, das unter dem Namen LSCube auf dem japanischen Markt angeboten wurde, übernommen. LSCube bestand bereits als Version 2.0 und wurde bei der Übernahme zu Enterprise Solution Builder, Version 3.0. In der Zwischenzeit sind Änderungen und Erweiterungen am Produkt vorgenommen wurden, so erklärt sich die Versionsnummer 3.02.

Den Kern des Produktes stellt ein Server dar, der LotusScript interpretiert und ausführt. Damit wird die Verarbeitung vom Client auf den Server verlagert. Mit der Installation eines Verbindungsprogrammes, dem ESB-Enabler, wird damit ein beliebiger Browser oder Notes-Client in die Lage versetzt, Anwendungen auszuführen, ohne dafür spezielle Ressourcen zur Verfügung stellen zu müssen.

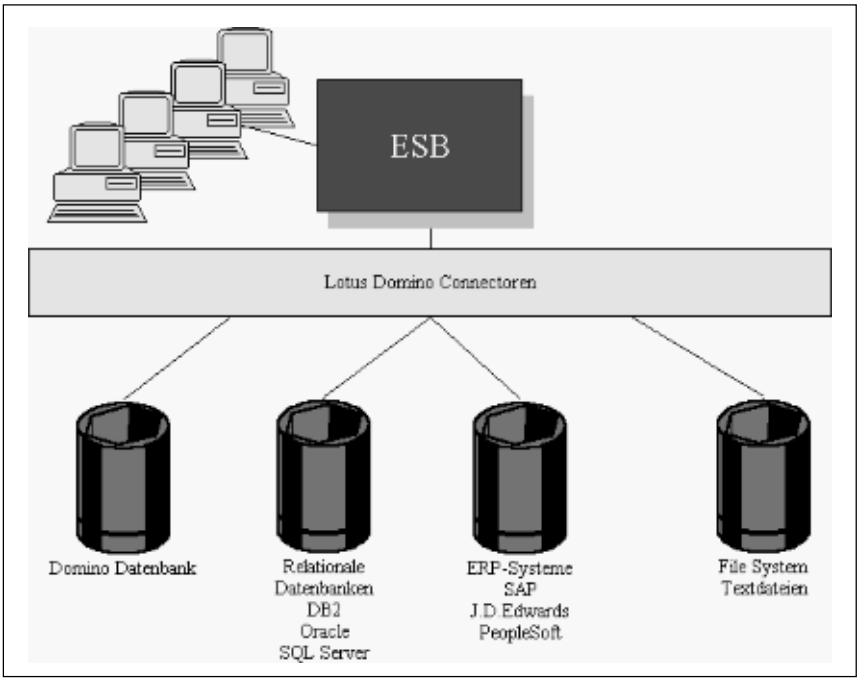


ESB Client-/Server-Anwendung

Der erste Schritt bedeutet lediglich eine Verlagerung der Programm-Ausführung auf den Server. Der ausgelagerte Teil kann keine Anwenderschnittstelle beinhalten und beschränkt sich deshalb auf Abwicklungsmodule. Die Präsentationsgrafik und die Interaktion

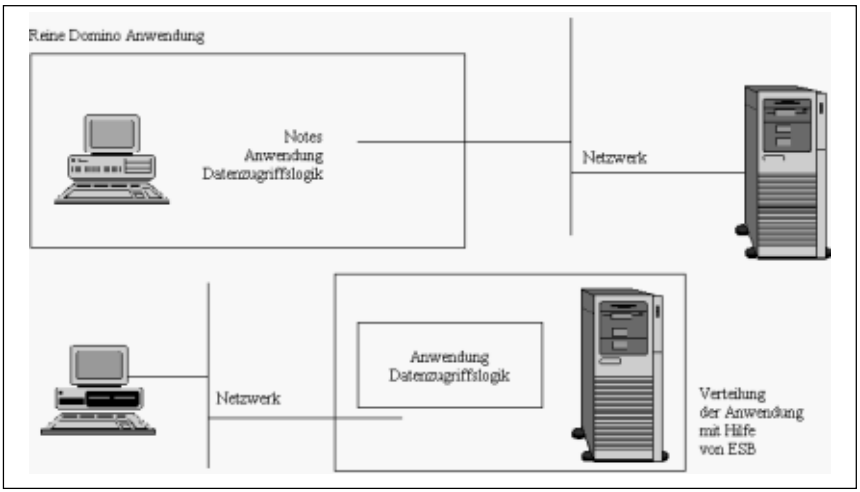
mit dem Benutzer werden vom Browser oder Notes Client übernommen.

Eine Anwendung lebt von ihren Daten und dem Zugriff auf die Daten. Eine reine Domino-Anwendung kann direkt auf Domino-Datenbanken auf dem Client oder dem Server zugreifen. Zusätzlich kann über die Lotus Domino Konnektoren auf externe Datenquellen zugegriffen werden. Der ESB verwendet die Lotus Domino Konnektoren und greift darüber auf die unterschiedlichsten Datenbanken zu. Eine davon kann eine Lotus Domino-Datenbank sein.



ESB-Datenbanken

Bereits über den Domino Server ist es möglich, auf verschiedene Datenbanken zuzugreifen. Auch die Anbindung von Browsern als Client ist möglich. ESB unterscheidet sich von der reinen Domino-Lösung dadurch, dass der Datenzugriff auf die verschiedenen Datenbanken wesentlich schneller erfolgt. Außerdem wird die Anwendungslogik vom Notes Client auf den Server verlagert. Das führt dazu, dass Anwendungen mit wenigen Änderungen auf den unterschiedlichsten Plattformen und unter Einsatz wechselnder Datenquellen eingesetzt werden können.

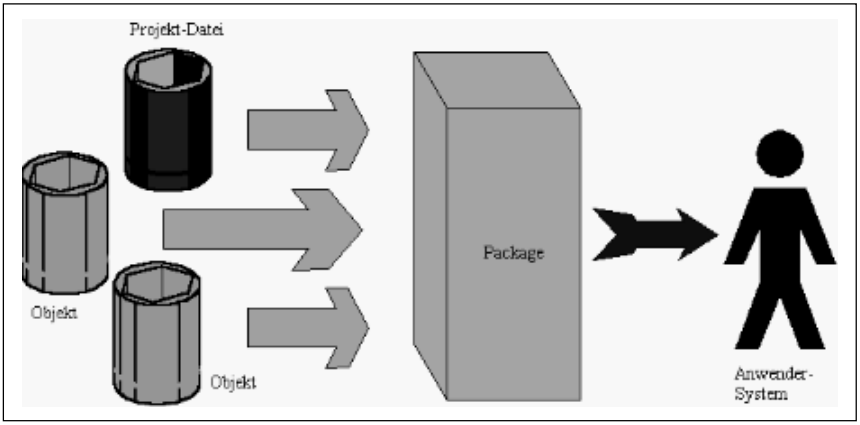


Verteilung der Anwendungslogik bei Domino und ESB

Über die reine Anwendungslogik hinaus bietet ESB eine eigene integrierte Entwicklungsumgebung für LotusScript-Anwendungen. Die LotusScript-Sprache wurde um spezielle Klassen für die Anbindung der Clients und den Zugriff auf Datenbanken erweitert. In der Version 3.02 wurde versucht, die Integration in die

Domino-Welt zu verstärken. Die Debugger von ESB und Domino Designer arbeiten eng zusammen.

Im Gegensatz zum Domino Designer liefert ESB die Möglichkeit, Programm-Module (Textdateien mit der Endung „.lss“) zu kompilieren und als Objekte auszuliefern bzw. wieder zu verwenden. Die fertige Anwendung wird zu einem „Package“ zusammengestellt und dann an die Anwender verteilt.



ESB-Programmdateien

Auf dem Anwendersystem muss der ESB-Server installiert sein, der dann die Programmausführung und den Datenbankzugriff steuert.

Für die Definition der Datenbankzugriffe enthält der Enterprise Solution Builder ein spezielles Werkzeug: den Query-Builder. Der Query-Builder stellt dem Entwickler die verschiedenen Datenquellen als Tabellen zur Verfügung. Er kann nun Tabellen und An-

sichten auswählen, die zu verwendenden Felder bestimmen und schließlich ein „SELECT“-Kommando erstellen lassen. Damit das Ergebnis dieses Kommandos auch wirklich die gewünschten Daten liefert, kann jederzeit in einer Vorschau angezeigt werden, welches Datenergebnis die ausgewählten Elemente erzielen würde. Zusätzlich können in diesem Umfeld auch direkt Datenmanipulationen an den Datenquellen vorgenommen werden. Neben dem „SELECT“-Kommando wird im Rahmen der Entwicklungsumgebung auch gleich der Quellcode für den Datenzugriff als LotusScript-Objekt erzeugt.

Im Rahmen der „Advanced Services“ werden Klassen mitgeliefert, die den gewohnten LotusScript-Umfang um Werkzeuge erweitern, die im Kommunikations- und Datenbankumfeld die Arbeit erleichtern:

- GetContext und SsContext

  zur Abfrage von Eigenschaften der aktuellen Sitzung (z.B. Eigenschaften der aktuellen Datenquelle)

- SsTimer

  ein ESB-eigener Trigger, der zeitgesteuerte Anwendungen erlaubt

- SsSharedStorage

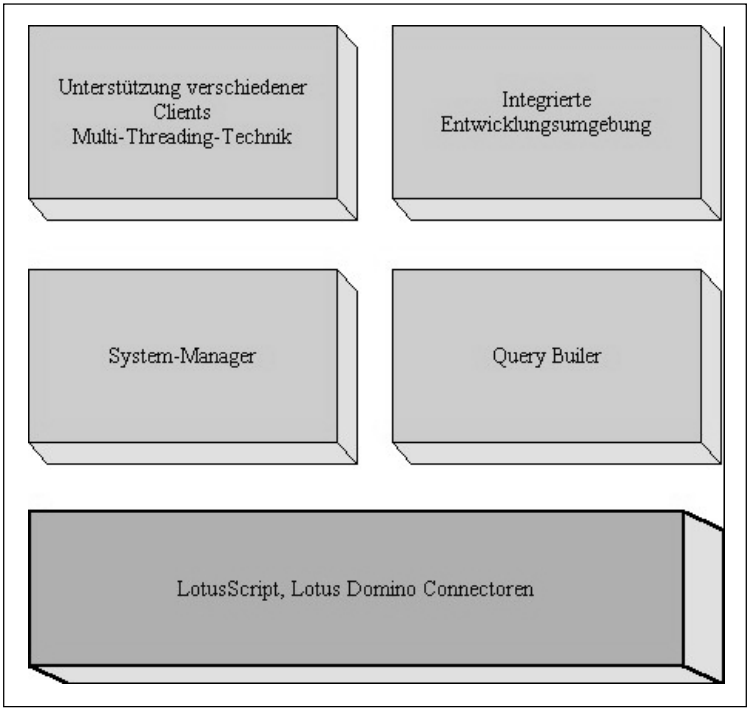
  Übergabe von Variablen an andere Programm-Module über gemeinsame Speicherbereiche

- SsList

  erweiterte Listen für verschiedene Datentypen

Wird ESB als Anwendungs-Server eingesetzt, so können verschiedene Tools zu seiner Steuerung genutzt werden. Der ESB-System-Manager liefert detaillierte Informationen über die laufenden Projekte, deren Status und Benutzer. Die Informationen können gefiltert und als Log-Datei ausgegeben werden. Dabei ist es möglich, den Umfang der Informationen so zu steuern, dass entweder alle Informationen, nur Warnungen oder nur Fehler ausgegeben werden.

Wenn man also den Leistungsumfang von ESB zusammenfassend darstellen will, ergibt sich etwa folgendes Bild:



ESB-Umfang

Damit ist ESB durchaus auch als Applikations-Server ohne Lotus Domino einsetzbar. Sollen nämlich kein Notes Client und keine Domino-Datenbanken verwendet werden, sondern z.B. eine Visual-Basic-Anwendung mit Daten einer DB2-Datenbank, so wird außer ESB kein Lotus Domino benötigt.

Zusammenfassend kann man sagen, dass es sich bei dem Enterprise Solution Builder um ein sehr interessantes Produkt handelt, das die Möglichkeiten der Anwendungsentwicklung rund um Lotus Domino erweitert und Server-orientierte Lösungen ermöglicht.

9

Entwicklung von Internetlösungen

Eine der Vorteile von Lotus Domino ist darin zu sehen, dass Programme und Daten in der gleichen Art in Datenbanken gespeichert sind. Dadurch gehört der Zugriff auf Daten zum Standard des Domino-Servers. Es wird keine zusätzliche Datenbank benötigt, um über den HTTP-Prozess auf Daten zuzugreifen und sie dem Browser-Benutzer anzuzeigen. Deshalb hat Lotus schon frühzeitig damit begonnen, den Domino-Server internetfähig zu machen.

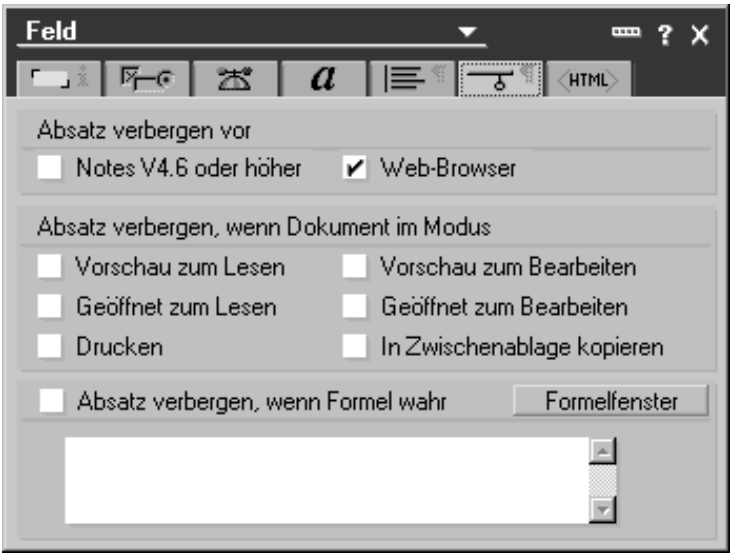
November 2001 / 7. EL

Sowohl von der Anwendungs- als auch von der Entwicklerseite her stehen Werkzeuge zur Verfügung, die dazu dienen, Domino-Anwendungen über Browser einzusetzen. Dabei sind die Standard-Einstellungen so getroffen, dass eine Notes-Datenbank ohne weitere Änderungen über den HTTP-Prozess als Internet-Anwendung und/oder -Datensammlung zur Verfügung steht. Da Domino-Datenbanken von Haus aus unstrukturierte Daten – also Textdokumente – verwalten können, ist es sehr einfach, auf diese Weise Texte zu erstellen und für das Internet verfügbar zu machen.

Ein weiterer Vorteil von Lotus Domino als Web-Server besteht darin, dass über die Replikation speziell geschützte Server im Firmen-Bereich zur Erstellung und Pflege der Daten genutzt werden können. Die Darstellung im Internet übernimmt dann ein Server in weniger geschützten Bereichen, der lediglich Repliken der Originaldatenbanken enthält.

Der Entwickler findet eine ganze Reihe von Unterstützungstools in den Datenbanken und kann damit Anwendungen entwickeln, die sowohl im klassischen Notes-/Domino-Umfeld als auch im Internet einsetzbar sind. Die ideale Lösung wäre es also, wenn einfach alle Gestaltungselemente des Domino-Designers automatisch in Browsern in der gleichen Art angezeigt würden, wie in einem Notes-Client. Leider arbeiten die Browser aber nicht in allen Bereichen exakt so, wie es die Notes-Software tun würde. Deshalb bestehen bereits in der Darstellung der Seiten erhebliche Unterschiede. Auch die Anpassung der Werkzeuge ist bei weitem nicht so weit fortgeschritten, wie es auf den ersten Blick erscheint. Erst im so genannten Rnext, also der Notes- und Domino-Version, die im Jahr 2002 erscheinen soll, sind Dinge wie das Rendern von HTML, die Umsetzung von Javascript und der Einsatz von

DHTML vorgesehen. In den bisherigen Versionen muss ständig zwischen Elementen für Notes und denen für das Internet umgeschaltet werden.



Gestaltungselemente im Browser verbergen

Dadurch ist es oft einfacher, Anwendungen für das Internet mit den entsprechenden Werkzeugen und die Notes-Anwendung mit dem Domino-Designer zu entwickeln.

Diese Seite ist zur Zeit nicht belegt.

10

Tools

Neben den Standard-Datenbanken mit Vorlagen und Anwendungen liefert Lotus eine Reihe zusätzlicher Hilfsmittel und Anwendungen, die die Möglichkeiten der Arbeit mit Lotus Domino erweitern und erleichtern. Es handelt sich dabei entweder um APIs oder um eigenständige Programme. Diese Tools können zum Lieferumfang der Standardsoftware gehören, werden zum Teil aber auch als eigenständige Software-Pakete verkauft (wie z.B. Domino Workflow).

November 2001 / 7. EL

Diese Seite ist zur Zeit nicht belegt.

10.1 DGW – Domino Global Workbench

Es war der Turmbau zu Babel, der nach der Bibel den Menschen die verschiedenen Sprachen und damit eine Menge Ärger einbrachte. Während viele Menschen sich an ihrer eigenen Sprache freuen, haben Software-Entwickler das große Problem, jedem Benutzer in seiner Sprache entgegenzutreten. Masken, Ansichten, Dialogboxen und andere Gestaltungselemente müssen so organisiert werden, dass jeder Benutzer seine eigene Sprache sehen und verwenden kann. Lotus bietet für die Domino-Datenbanken ein Hilfsprogramm an, das den Entwickler bei der Lösung dieses Problems unterstützt. Das Programm hat den Namen „Domino Global Workbench (DGW)" und befindet sich auf der Programm-CD, die die Designer-Software enthält.

Die Domino Global Workbench wirkt auf den ersten Blick unübersichtlich und erfordert eine gewisse Einarbeitungszeit, bis man die Datenbanken in der richtigen Form benannt und sich an die Vorgehensweise der DGW gewöhnt hat. Wichtig ist, dass man sich intensiv mit dem Einsatz der richtigen Alias-Namen auseinandersetzt. Wenn die Einarbeitungszeit überstanden ist, erweist sich die DGW als sehr gutes Werkzeug für die Sprachanpassung von Domino-Anwendungen, die – wie man das von Lotus Domino erwartet – ganz stark auf die Zusammenarbeit im Team ausgerichtet ist.

November 2001 / 7. EL

Diese Seite ist zur Zeit nicht belegt.

10.1.1 Die Installation der DGW

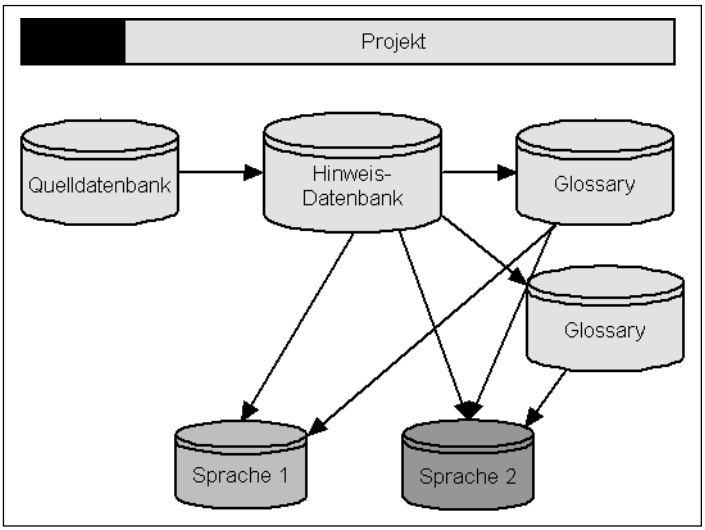
Im Verzeichnis „Apps" der Designer-Programm-CD befindet sich das Verzeichnis „DGW". Von dort kann das Programm „SetUp.exe" gestartet werden. Wie von Lotus gewohnt, wird die Installation von Dialogen begleitet, die alle relevanten Daten abfragen und die Dateien auf dem Rechner installieren. Je nach Art der Notes-Installation auf der Arbeitsstation muss das Notes-Verzeichnis noch unter Windows als Pfad hinterlegt werden.

November 2001 / 7. EL

Diese Seite ist zur Zeit nicht belegt.

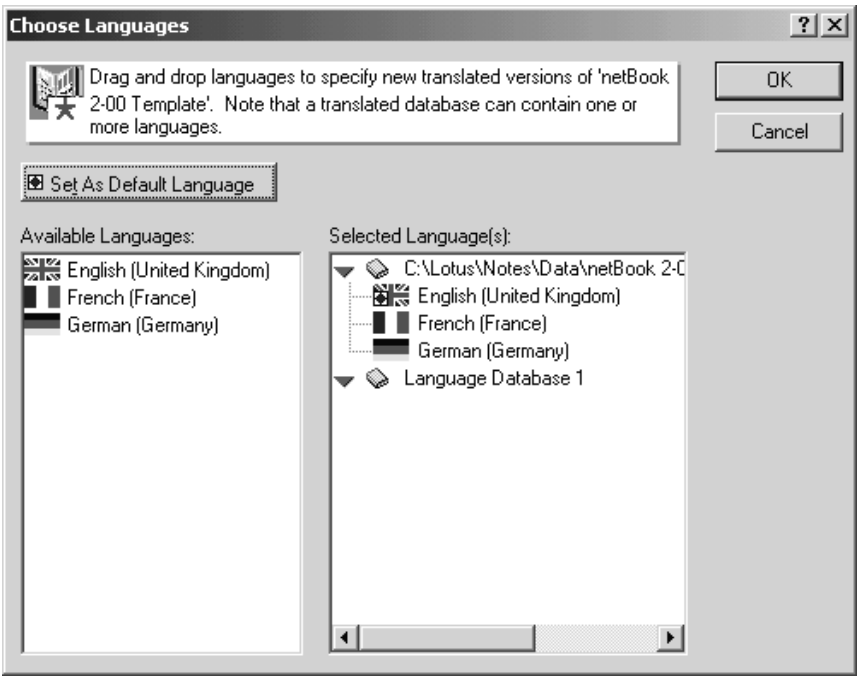
10.1.2 Aufbau und Konzeption

Die Domino Global Workbench verwendet Projekte, um Datenbanken und die dazugehörigen Hilfsmittel zu verwalten. Als Basis für ein Projekt dient eine Notes-Datenbank, die mehrere Sprachversionen erhalten soll. Um eigene Hinweise anbringen zu können, erstellt DGW eine Kopie der Ursprungsdatenbank. Dazu kommt eine Datenbank (Glossary), in der die einzelnen Sätze oder Worte zu finden sind, die übersetzt werden müssen, und schließlich mündet das Ergebnis in Datenbanken für die gewünschten Zielsprachen. Das System erlaubt es, nach einer Vorarbeit durch einen Entwickler, die Übersetzung durch Sprachspezialisten vornehmen zu lassen, die von Programmierung nichts oder nur wenig verstehen.



Projekt-Aufbau

Neben der Erstellung getrennter Sprachversionen ist auch die Erzeugung einer Datenbank möglich, die alle Gestaltungselemente in den verschiedenen Sprachen enthält. In diesem Fall wird lediglich die Sprache über die gewünschte Datenbank gezogen und damit ein entsprechender Verweis in die Projektdatenbank eingefügt.

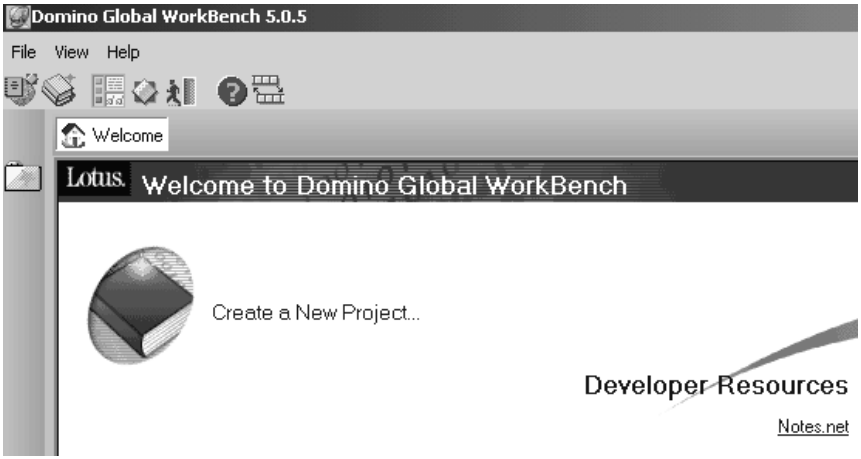


Sprache einer Datenbank zuordnen

Da bei diesem Prozess neben der Ursprungsdatenbank eine Reihe weiterer Datenbanken mit ähnlichen Bezeichnungen entstehen, sollte von Anfang an auf möglichst sprechende Datenbank-Titel geachtet werden, damit spätere Verwechslungen ausgeschlossen sind.

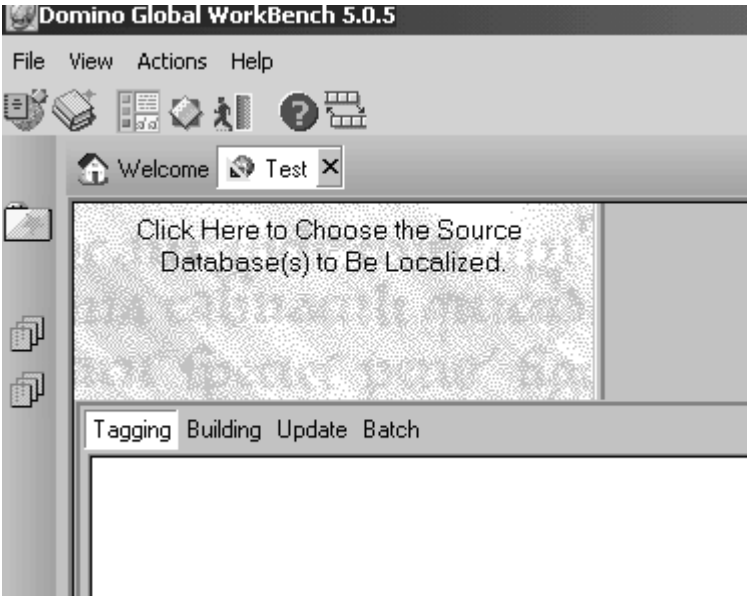
10.1.3 Ein neues Projekt

Wird DGW zum ersten Mal gestartet, ähnelt das Startbild dem des Domino-Designers. Auch hier wird eine Willkommen-Seite gezeigt, die es ermöglicht, direkt ein Projekt zu erstellen.



Die Willkommen-Seite der DGW

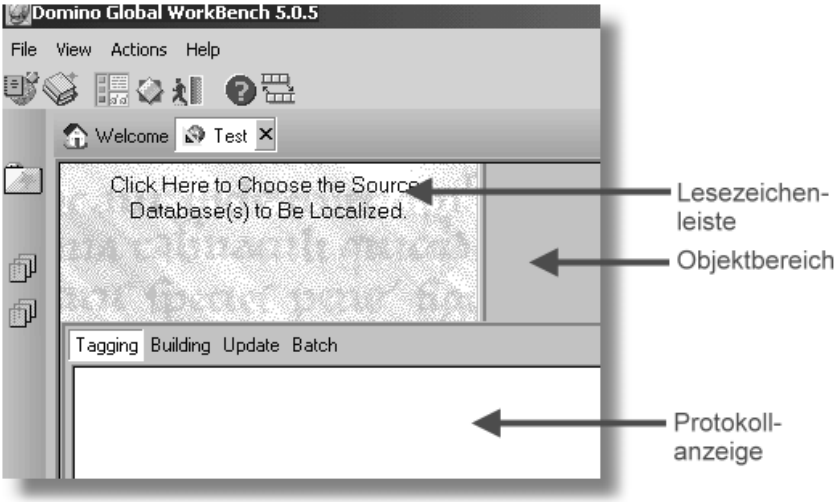
Auch die folgende Einteilung ist an den Designer angelehnt. Aber hier zeigen sich auch die ersten Unterschiede. In der Lesezeichenseite ist eine Textzeile eingeblendet, die die Angabe der Ausgangsdatenbank auslöst.



Angabe der Ausgangsdatenbank

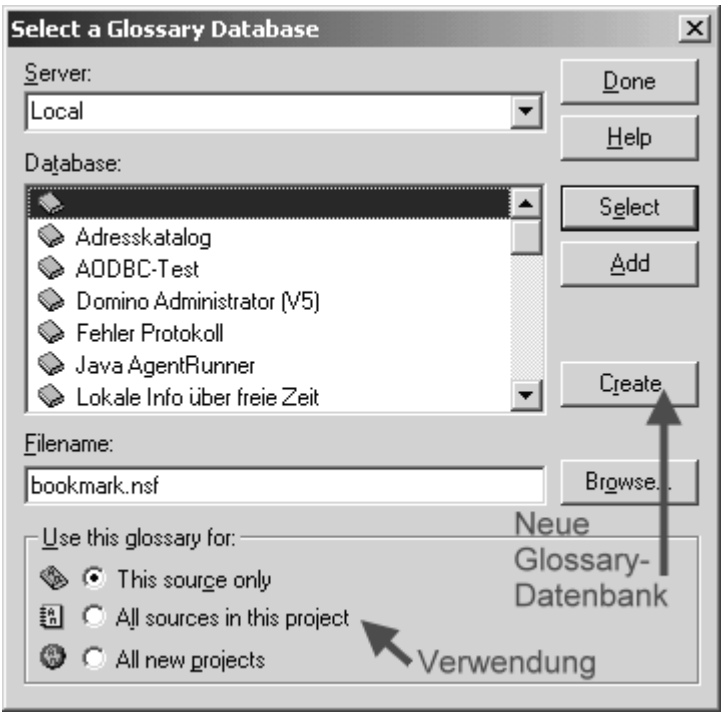
Der Arbeitsbildschirm ist in drei Arbeitszonen eingeteilt:

- die Lesezeichenseite mit den verschiedenen Datenbanken,
- den Objektbereich, der die eigentlichen Arbeitsbildschirme enthält,
- und den Protokollbereich, in dem die verschiedenen Arbeitsschritte angezeigt werden, die die DGW bei ihren Aktionen durchläuft. Dazu kommen Warnungshinweise und Fehlermeldungen.



Die Arbeitsbereiche der DGW

Wird die Hinweiszeile für die Ausgangsdatenbank angeklickt, so erscheint ein Auswahldialog, über den die Datenbank gefunden werden kann. Die neue Datenbank wird in die Lesezeichenleiste übernommen und der nächste Text mit der Aufforderung, eine Glossary-Datenbank auszuwählen, erscheint darunter. Im darauf folgenden Dateidialog findet sich eine Schaltfläche für die Anlage einer neuen Glossary-Datenbank.

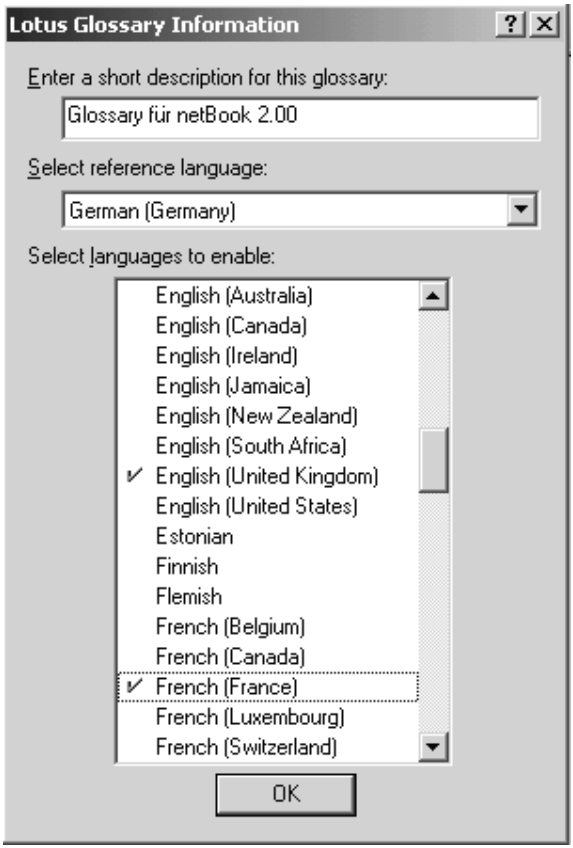


Neue Glossary-Datenbank

Zusätzlich kann angegeben werden, ob in dieser neuen Datenbank Begriffe abgelegt werden sollen, die Verwendung finden in

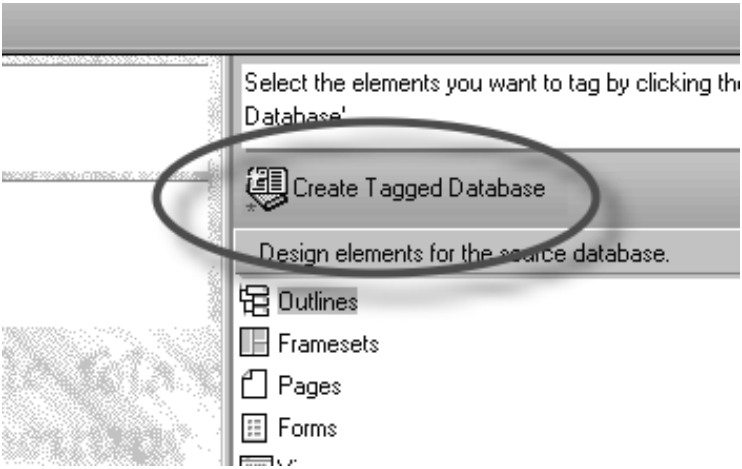
- der aktuellen Ausgangsdatenbank
- allen Datenbanken des Projektes
- allen weiteren Projekten.

Damit die Datenbank für die Übersetzungsbegriffe endgültig angelegt werden kann, sind jetzt noch die Sprachen anzugeben, die verwendet werden sollen. Dabei wird unterschieden zwischen der Referenz-Sprache (also der Ausgangssprache) und den Zielsprachen.



Glossary-Sprachen festlegen

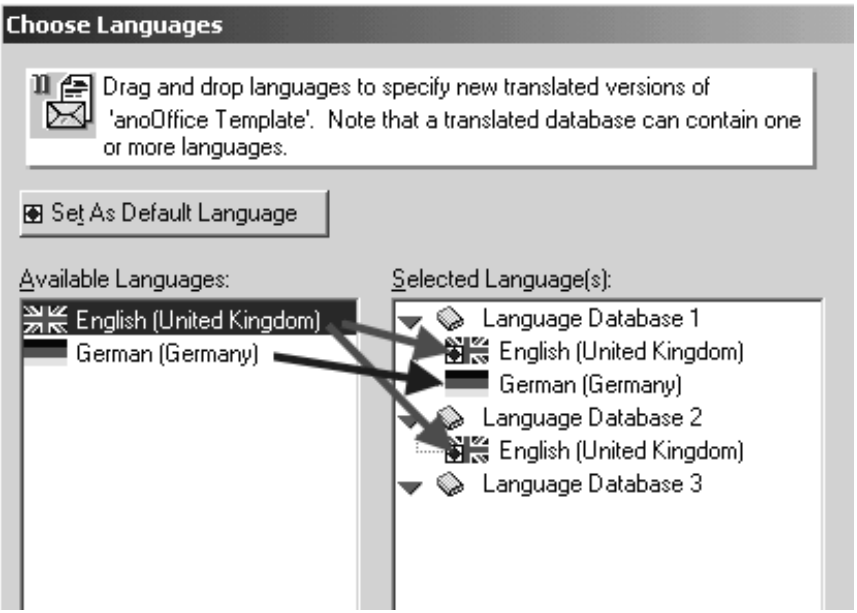
Als Nächstes wird eine Kopie der Ausgangsdatenbank angelegt, in der die DGW ihre Vermerke und Suchschlüssel hinterlegt (die „Tagged"-Datenbank).



Tagged-Datenbank anlegen

In der Tagged-Datenbank sind jetzt alle Gestaltungselemente in der Referenz-Sprache mit den entsprechenden Suchschlüsseln vorhanden. Damit die sprachbezogenen Gestaltungselemente erstellt werden können, werden jetzt die Sprachen ausgewählt, die dafür in Frage kommen.

Soll z.B. eine Datenbank für die Verwendung in deutscher und englischer Sprache und eine weitere für die Verwendung in englischer Sprache erstellt werden, so erfolgt die Sprachzuweisung wie in unten dargestellt.



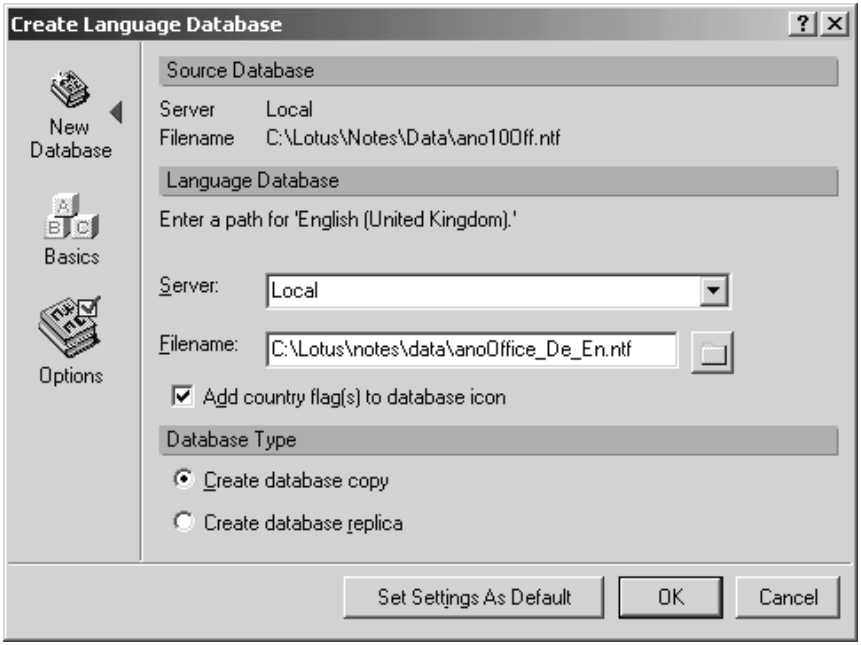
Zieldatenbank in verschiedenen Sprachkombinationen

❶❷❸ Ausführungsschritte:

Die Erstellung der Sprachdatenbanken erfolgt in den Schritten:

1. Sprachdatenbank auswählen.
2. Erstellen des Dateinamens sowie Optionen im Folgedialog festlegen.
3. DGW erstellt die Sprachdatenbank und protokolliert im unteren Bereich des Bildschirmes, welche Elemente in die Glossary-Datenbank übernommen wurden und ob Fehler dabei aufgetreten sind.

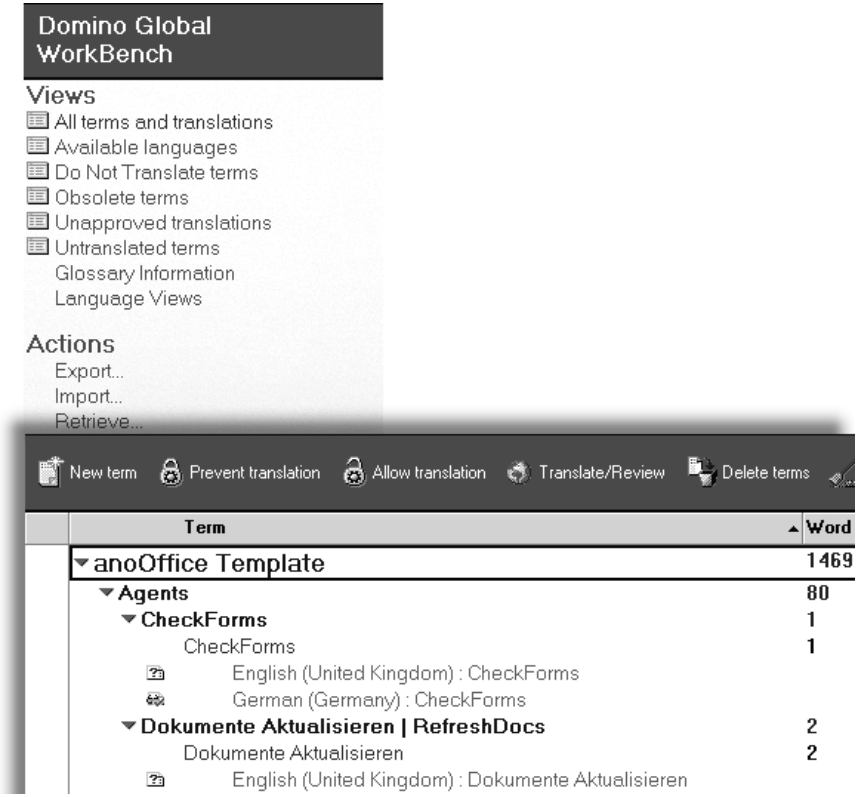
❶❷❸



Sprachdatenbank erstellen

Diese Schritte werden für jede Datenbank durchgeführt. Damit ist die Glossary-Datenbank für die Übersetzung vorbereitet.

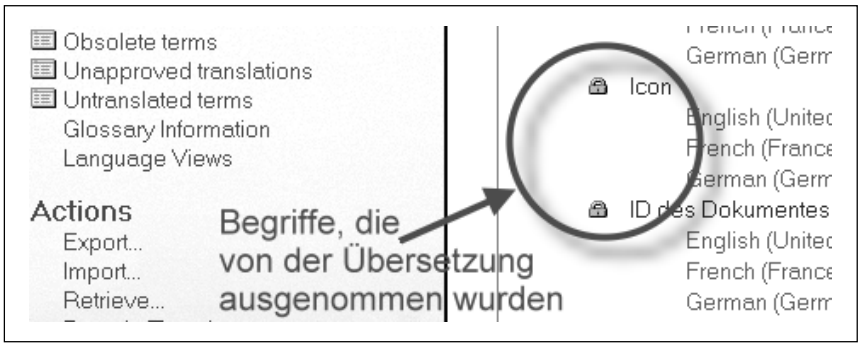
Sie hat jetzt folgendes Aussehen:



Die Glossary-Datenbank

Es handelt sich bei der Glossary-Datenbank um eine normale Domino-Datenbank, die über den Notes-Client gestartet wird. Im ersten Durchlauf enthält die Glossary-Datenbank viele Begriffe, die bei näherem Hinsehen für die Übersetzung überhaupt nicht benötigt werden. Das sind z.B. Kommentare in Masken, Namen

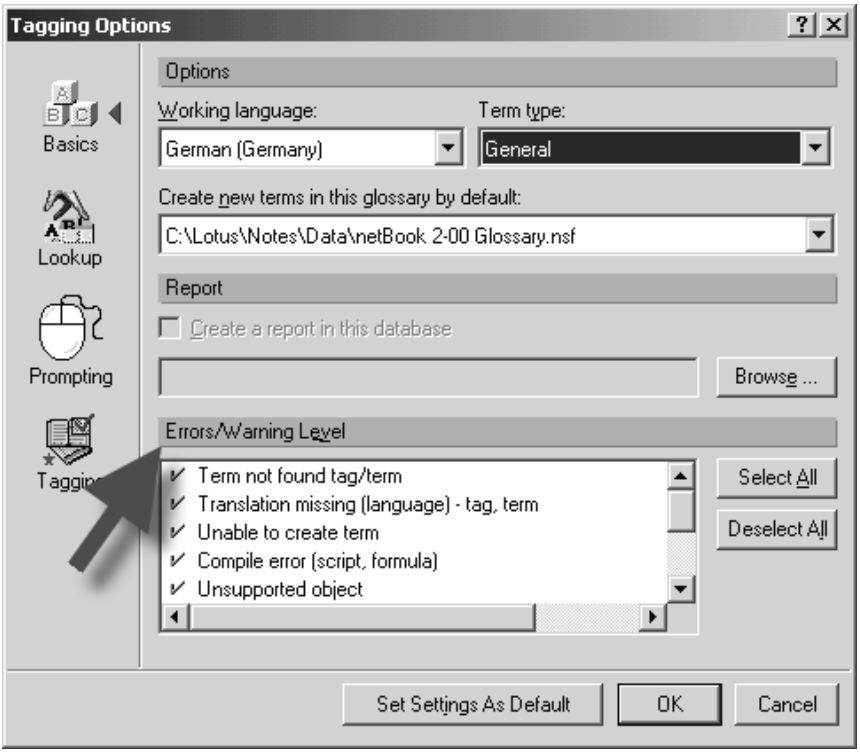
von Masken und Ansichten usw. Diese Begriffe können jetzt einzeln gekennzeichnet und von der Übersetzung ausgeschlossen werden. Dazu wird der Begriff markiert und die Schaltfläche „Prevent Translation" angeklickt.



Begriffe von der Übersetzung ausnehmen

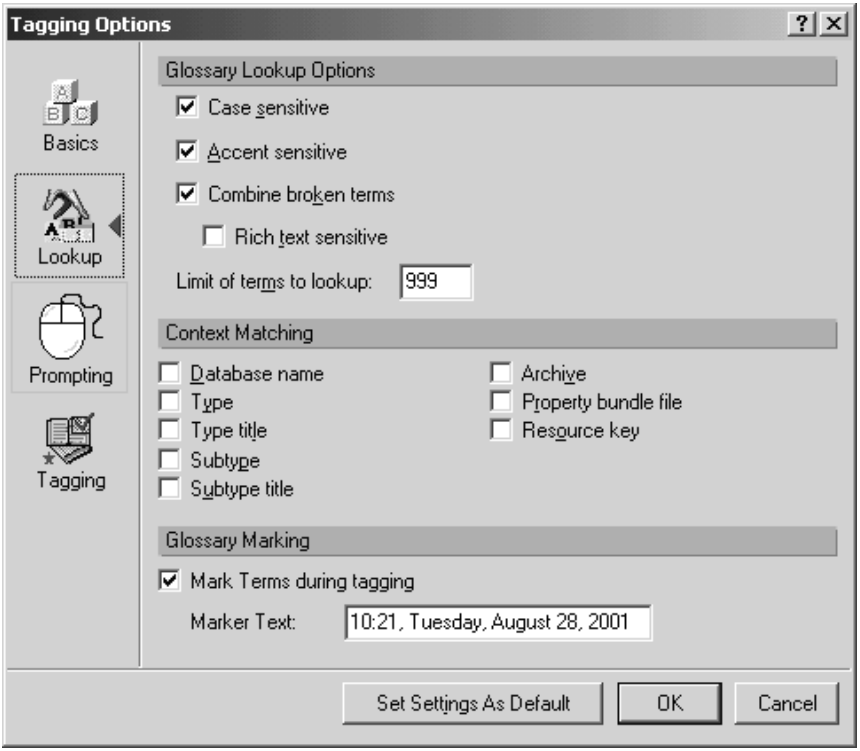
Dieses Verfahren ist natürlich mühsam und lässt sich mit Hilfe einer Option der DGW erheblich vereinfachen. Dazu wird im Menü „Actions" unter „Tagging Options" festgelegt, welche Gestaltungselemente unter welchen Bedingungen in die Übersetzung einbezogen werden sollen.

Bei den Basis-Angaben kann hier eingestellt werden, welche Warnhinweise und Fehlermeldungen erzeugt werden sollen.



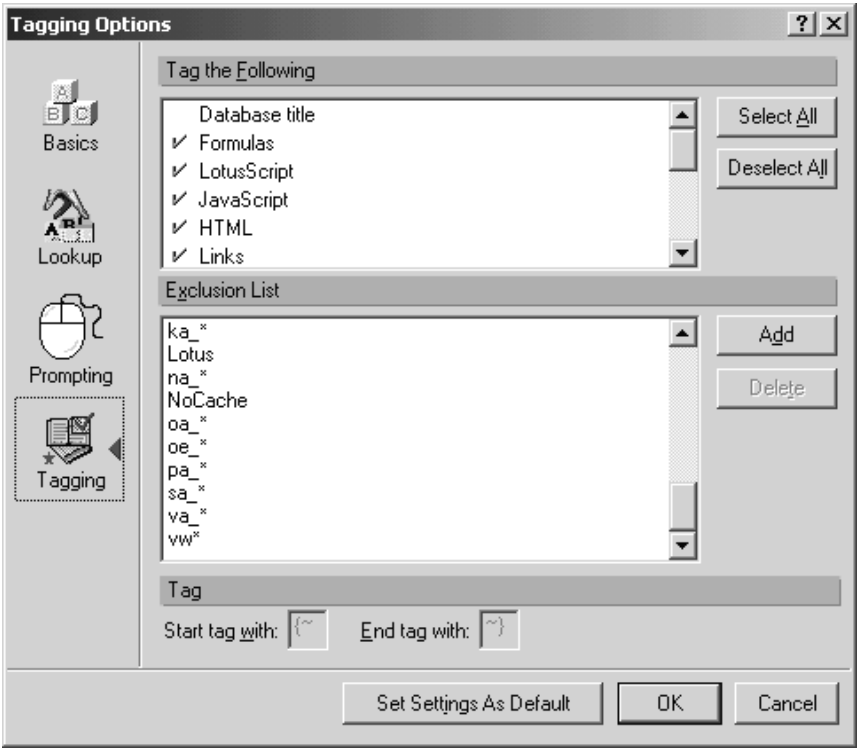
Tagging Options – Basis

Im Bereich „LookUp" wird festgelegt, auf welche Art die Begriffe in der Datenbank gesucht und mit dem Glossary verglichen werden.



agging Options – Lookup

Unter „Prompting" kann eingestellt werden, ob bestimmte Vorkommnisse zu einer Abfrage am Bildschirm führen sollen. Die wichtigste Funktion verbirgt sich hinter „Tagging". Hier wird festgelegt, welche Textarten durchsucht werden und welche Begriffe nicht übersetzt werden sollen.



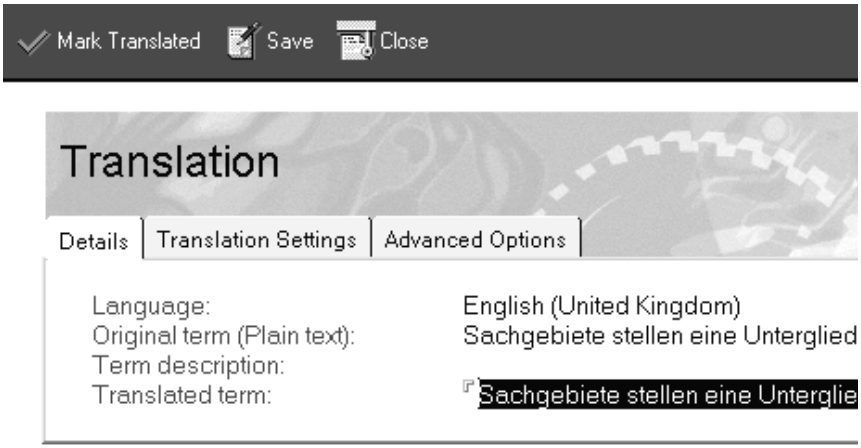
Tagging Options – Tagging

Dabei zeigt sich, dass die Namensgebung von Ansichten, Masken und Feldern von großer Bedeutung für die Verwendung der DGW ist. Solange die Namen mit Kurzformen gekennzeichnet sind, die üblicherweise in Wörtern nicht vorkommen, kann einfach über einen Wildcard-Begriff ein Ausschluss definiert werden (z.B. frm*). Handelt es sich aber um Buchstabenkombinationen, die auch in anderem Zusammenhang möglich sind (z.B. fra*), dann ist dieser Weg versperrt. Lotus schlägt deshalb vor, generell eine Kombination von Buchstabenkürzel und dem Zeichen "_" zu verwenden (z.B. pa_*). Dabei wird natürlich auch unterstellt, dass bei allen Designelementen mit Alias-Namen gearbeitet wird.

Diese Seite ist zur Zeit nicht belegt.

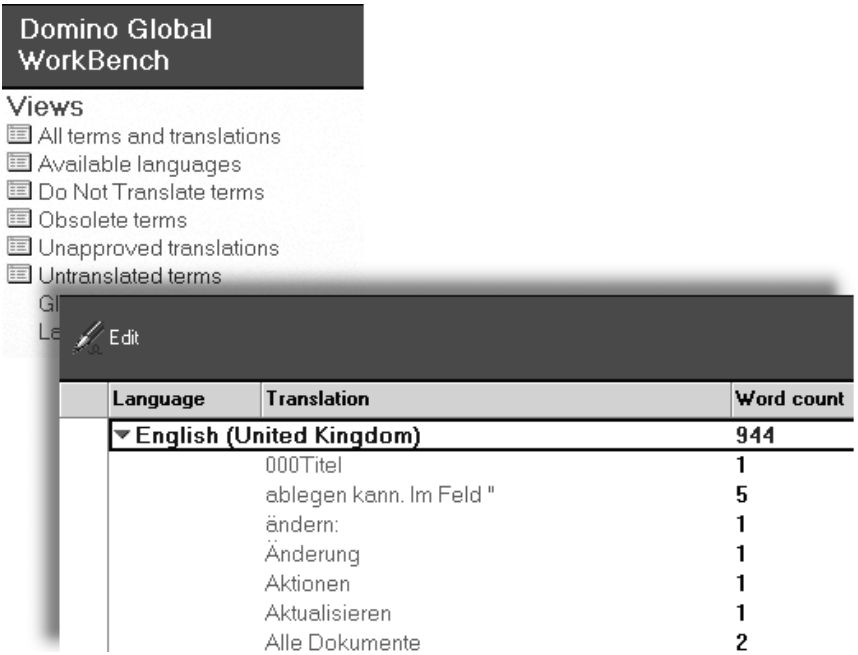
10.1.4 Die Übersetzung

Sind alle Vorbereitungen getroffen, so kann die eigentliche Übersetzungsarbeit beginnen. Die Übersetzung kann entweder direkt in der Glossary-Datenbank über exportierte Textdateien vorgenommen werden. Soll ein Begriff direkt übersetzt werden, so wird das entsprechende Sprachdokument geöffnet und der übersetzte Begriff in das Feld „Translated Term" eingetragen. Ist der Begriff damit endgültig übersetzt, so wird er mit Hilfe der Schaltfläche „Mark Translated" entsprechend gekennzeichnet.



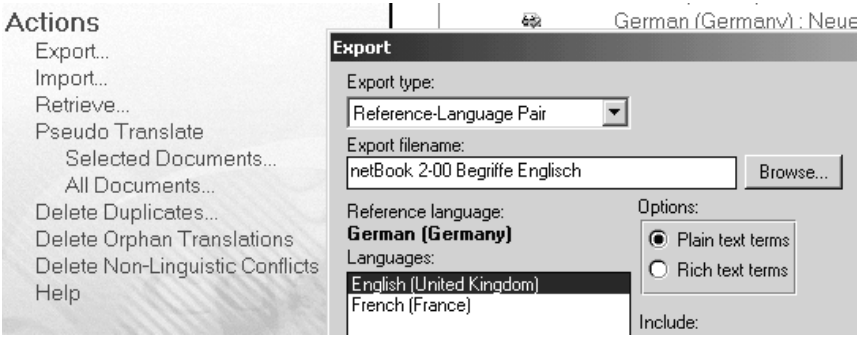
Begriffe in der Glossary-Datenbank übersetzen

Der Stand der Übersetzung lässt sich in der Ansicht „Untranslated Terms" ablesen. Hier werden alle Begriffe aufgelistet, für die noch keine Übersetzung vorliegt.



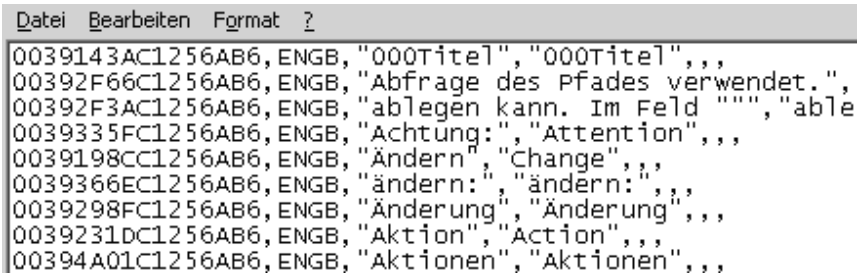
Die Ansicht „Untranslated Terms"

Häufig müssen Übersetzungen von externen Personen durchgeführt werden, die nicht die Möglichkeit haben, mit Domino-Datenbanken zu arbeiten. In diesem Fall kann eine einfache Textdatei mit den Begriffen ausgegeben werden.



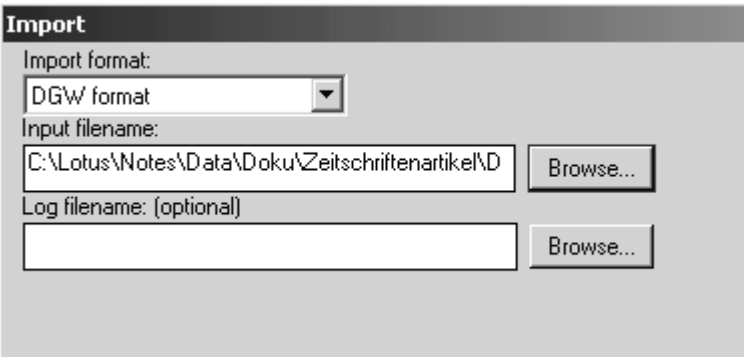
Begriffe exportieren

Dabei werden Sprache und Ausgabedatei definiert und die Aktion „Exportieren" aufgerufen. Die DGW erstellt dann eine Textdatei mit je einer Zeile für einen Begriff. Die Zeilen sind mit einem Erkennungsschlüssel ausgestattet, der den Re-Import der Begriffe ermöglicht.



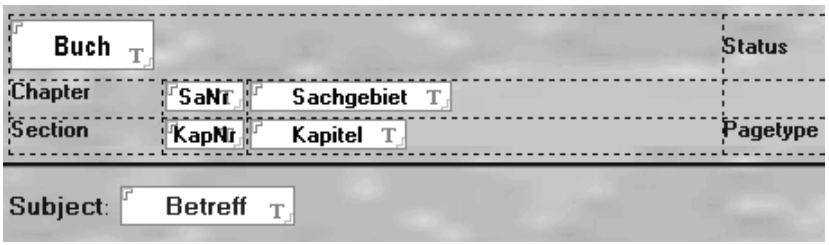
Textdatei mit Begriffen

Der Übersetzer muss jetzt lediglich den zweiten Begriff, der für die Sprachversion eingesetzt wurde, durch den übersetzten Begriff ersetzen. Danach kann diese Textdatei mit der Aktion „Importieren" wieder übernommen werden.



Begriffe importieren

Die Sprachdatenbanken werden aktualisiert mit „Re-create Tagged Database" und „Re-build Language Database". In den Sprachdatenbanken werden jetzt die nicht ausgeschlossenen Begriffe aus der Glossary-Datenbank in die verschiedenen Gestaltungselemente übernommen.



Übersetzte Maske im Designer

10.1.5 Sprachsynchronisation

Zur Unterstützung des Anpassungsprozesses der Sprachdatenbanken an Änderungen in der Ausgangsdatenbank kann auf die Synchronisations-Funktion der DGW zurückgegriffen werden. Es handelt sich dabei um ein System aus Masken-Anpassungen, Agenten und einem Server-Prozess, der die Überwachung der Änderungen übernimmt und für eine automatische Anpassung aller abhängigen Datenbanken sorgt.

November 2001 / 7. EL

Diese Seite ist zur Zeit nicht belegt.